VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Content Management System s využitím HTML5

Content Management System Based on HTML5

2013 Jan Janoušek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jan Janoušek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Content Management System s využitím HTML5
Content Management System Based on HTML5

Zásady pro vypracování:

Cílem práce je implementace Content Management Systému založeném na standardu HTML5. V rámci práce budou představeny hlavní přednosti standardu, které budou prezentovány na dílčích komponentách systému (např. podpora akcelerované grafiky, apod.). Na těchto komponentách budou provedeny výkonnostní experimenty. Nad celým systémem budou provedeny UX (User Experience) testy.

Body zadání.
1. Představení HTML5 standardu s ohledem jeho přidanou hodnotu oproti existujícím řešením.
2. Návrh a implementace Content Management Systému.
3. Návrh a implementace komponent, na kterých budou prezentovány přednosti HTML5 (např. akcelerovaná grafika, dotykové rozhraní, apod.).
4. Provedení výkonnostních experimentů navržených komponent, UX experimentů.
5. Závěrečné zhodnocení práce a reálné nasazení systému.

Seznam doporučené odborné literatury:

[1] M. Pilgrim: HTML5 - Up and Running, 2010, ISBN:0596806027, 1st edition, O'Reilly Media, Inc.
[2] Steve Fulton and Jeff Fulton: HTML5 Canvas - Native Interactivity and Animation for the Web, 2011, ISBN: 978-1-449-39390-8, I-XIX, 1-628, O'Reilly

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Radomír Mika**

Konzultant diplomové práce: Ing. Petr Gajdoš, Ph.D.

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2013

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.*

V Ostravě 18. dubna 2013 .....

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 18. dubna 2013 .....

Rad bych poděkoval panu Ing. Petru Gajdošovi, Ph.D., za odborné vedení a rady při zpracovávání této práce.

Abstrakt

Cílem této diplomové práce je vytvoření content management systému s využitím moderních technologií z rodiny HTML5 a demonstrace jejich možností. První část této práce se zabývá představením samotného standardu HTML5, jakožto klíčové technologie pro tuto diplomovou práci a jeho srovnáním s jinými technologiemi.
Druhá část práce se zabývá návrhem a implementací komponent na nichž bude systém postaven a také praktickou ukázkou možností hardwarově akcelerované grafiky na webu, jakožto nedílné součásti standardu HTML5.
Třetí část práce se věnuje návrhu a implementaci klíčových částí systému.
Poslední část této práce se věnuje user experience testování navrženého systému a výkonnostnímu testování komponenty využívající hardwarově akcelerovanou grafiku.

Klíčová slova: HTML, HTML5, CMS, JavaScript, WebGL, akcelerovaná grafika, vizualizace grafů

Abstract

The main goal of my thesis is to create content management system based on modern technologies from HTML5 family and to show all their capabilities. The first part deals with introduction the HTML5 standard itself as the key technology for this thesis and his comparison with other technologies.
The second part deals with design and implementation of components on which our system will be build and also with visual demonstration of the possibilities that hardware accelerated graphics have on the web as inseparable part of standard of HTML5.
The third part of my thesis is dedicated to design and implementation of the key parts of system.
And finally the last part is dedicated to user experience testing of propounded system and performance testing of component using hardware accelerated graphics.

Keywords: HTML, HTML5, CMS, JavaScript, WebGL, accelerated graphics, graph visualisation

Seznam použitých zkratek a symbolů

HTML	–	Hyper Text Markup Language
W3C	–	World Wide Web Consortium
JS	–	JavaScript
CSS	–	Cascading Style Sheets
CSS3	–	Cascading Style Sheets Level 3
MVC	–	Model View Controller
MVC 3	–	Active Server Pages .NET Model View Controller 3
RIA	–	Rich Internet application
MXML	–	Magic eXtensible Markup Language
XAML	–	Extensible Application Markup Language
DOM	–	Document Object Model
TLS	–	Transport Layer Security
XML	–	Extensible Markup Language
dpi	–	Dots per inch
dpcm	–	Dots per centimeter
px	–	Pixel
XAML	–	Extensible Application Markup Language
HTTP	–	Hypertext Transfer Protocol
CSFR	–	Cross-site request forgery
SVG	–	Scalable Vector Graphics
API	–	Application programming interface
OpenGL	–	Open Graphics Library
GLSL	–	Open Graphics Library Shading Language
CMS	–	Content management system
WYSIWYG	–	What You See Is What You Get
MIME	–	Multipurpose Internet Mail Extensions
XHR2	–	XMLHttpRequest 2
UX	–	User Experience

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Od prosince roku 1990, kdy Timothy John Berners-Lee, Robert Cailliau a skupinka jejich studentů stvořila první hypertextovou stránku, uběhlo už spoustu času a stihlo se toho spoustu udát. Původním cílem tvorby hypertextu (dnes již označovaného spíše jako web), bylo vytvořit organizovanou, navzájem provázanou skupinu odborných textů, které by byly přístupné odkudkoliv z internetu. Tato myšlenka víceméně zůstala až dodnes, až na odbornost většiny textů.

Díky tomu, že měl web sloužit odborné veřejnosti, vystačily si první weby jen se základním HTML, které navíc nebylo v době vzniku prvních hypertextových dokumentů ani standardizováno. Ovšem díky tomu, že se web začal velmi rychle rozšiřovat i mezi neodbornou veřejnost, začaly růst nároky na něj kladené. Hlavně na prezentační část. Vznikly proto technologie jako CSS a později i JavaScript.

Celé toto období je ovšem velmi chaotické, protože spousta věcí není stále standardizována a dostatečně zdokumentována. Vznikají tak různé webové prohlížeče, které se ovšem v podpoře a interpretaci jednotlivých dnes již standardů rozcházejí.

Toto období, kdy je tvorba webového obsahu ryze v rukou odborné veřejnosti se často označuje jako Web 1.0.

Časem ovšem vznikají myšlenky na to, že by obsah webu mohli tvořit i běžní uživatelé bez větších technických znalostí. Vznikají tak všemožné webové služby, jako jsou sociální sítě, blogy, fotogalerie, servery pro sdílení videa, souborů a spousta dalších. Tento rozkvět webu je ovšem možný jen díky novým technologiím, které přinášejí stále nové možnosti pro vývojáře. Toto období, které trvá v podstatě až dodnes, se často označuje jako Web 2.0.

Aktuální trendy ukazují na to, že bude význam webu stále narůstat. Spousta firem se již dnes snaží přenést své aplikace na web. V podstatě všechny operační systémy dnes nabízejí nějaké provázání s webem. Ať už jde o synchronizaci, nebo dokonce přímý vývoj aplikací za pomoci webových technologií, jako Windows 8. To vše je ale možné jen díky rodině technologií HTML5.

Tato diplomová práce bude zaměřena hlavně na tyto nové technologie. V první části se pokusím přiblížit jednotlivé technologie společně s příklady jejich využití a to hlavně pro vývoj webových aplikací. Většinu těchto technologií, jak bylo již zmíněno, je možné využít i při vývoji desktopových aplikací pro Windows 8.

Druhá část této práce je zaměřena na vývoj Content Management Systému s výrazně odlišnou koncepcí od většiny běžně používaných systémů pro zprávu obsahu. Systém bude postaven na modernách technologiích, jako je ASP.NET MVC 3, SQL Server a HTML 5.

2 HTML 5

Standard HTML je tu s námi již od roku 1995, kdy byla publikována verze 2.0. Ve specifikaci této verze je jako popis technologie HTML uvedeno, že jde o jednoduchý značkovací jazyk, který slouží k vytváření platformně nezávislých hypertextových dokumentů [1]. Tento popis jen trochu jinak formulovaný přetrvával přes všechny následující verze (3.2, 4.0 a 4.01), až do příchodu standardu HTML 5. V aktuální verzi standardu je namísto toho uvedeno, že tato verze přichází s novými funkcemi pro tvůrce **webových aplikací**, nové prvky vycházející z praktických potřeb autorů webů a také jsou nově standardizovány požadavky na uživatelské klienty (tedy internetové prohlížeče).[2]

Již z tohoto popisu je patrné, že HTML 5 se vydává výrazně jiným směrem, než jeho předchůdci. Důležitou vlastností této verze je, že do ní přispívala a stále přispívá spousta významných firem z oboru, díky čemuž se celý standard daleko více než jeho předchůdci zaměřuje na **řešení ryze praktických problémů** v dané oblasti. A snaží se nejen **konkurovat dosavadním RIA technologiím**, ale dokonce těm desktopovým. Důkazem toho je například **možnost vyvíjet nativní aplikace pro různá mobilní zařízení a Windows 8**.

Je ovšem potřeba podotknout, že když se dnes mluví o HTML 5, tak nemluvíme jen o standardu *HyperText Markup Language 5*, ale o celé rodině technologií okolo tohoto standardu. Těmito technologiemi jsou Cascading Style Sheets 3, JavaScript >=1.7 a v neposlední řadě API poskytovaná prohlížečem.

2.1 Přínos HTML 5 oproti ostatním RIA technologiím

Již dlouhou dobu máme na trhu velké množství různých RIA technologií, jenž umožňují tvorbu webových aplikací a obecně interaktivního obsahu. Těmi nejznámějšími jsou Adobe Flash/Flex, Microsoft Silverlight a JavaFX. I když má každá z těchto technologií své výhody a nevýhody, tak ve výsledku jsou si svými možnostmi víceméně rovnocenné a je velmi obtížné rozhodnout, která z nich je ta nejlepší. Nabízí se tedy otázka, zda-li vůbec je potřeba nová RIA technologie v podobě HTML 5 a co nám tato technologie přinese navíc oproti těm stávajícím.

Nejprve je tedy potřeba udělat si základní přehled v těch stávajících. Tak aby bylo možné je srovnat s HTML 5.

2.1.1 Adobe Flash/Flex

Adobe Flash patří do skupiny technologií využívající pro svůj běh zásuvný modul prohlížeče. Dle nezávislých statistik ze začátku roku 2013 je tento plugin (Flash Player) nainstalován na 95.83% počítačů připojených k internetu [4]. Stává se tak jednoznačně nejlépe podporovanou RIA technologií využívající zásuvné moduly.

Pro vývoj flash aplikací lze využít dvou oficiálních nástrojů. Buď'to aplikace Adobe Flash, nebo Adobe Flex builder. Obě tyto aplikace jsou komerční a liší se od sebe hlavně svým zaměřením. Adobe Flash je zaměřen hlavně na grafiku a animace. Adobe Flex builder se naopak zaměřuje na vývoj aplikací. Pro vývoj těchto aplikací není ovšem nutné

využívat právě Flex Builder, ale stačí jakýkoliv textový editor. Zdrojové kódy následně stačí zkompilovat pomocí Flex SDK, jenž je open souce.

Programovací jazyk, jenž se při vývoji flash aplikací využívá, se nazývá Action Script (aktuálně ve verzi 3.0). Podobně jako u jazyku C# nebo Java je tento kód kompilován do mezikódu, jenž je následně interpretován Flash pluginem.

Kromě uvedených nástrojů existuje spousta více či méně kvalitních open source nástrojů, jenž umožňují flash aplikace vyvíjet. Těmi známějšími jsou například OpenLaszlo, nebo Google PlayN.

Tato technologie je podporována na velkém množství zařízení, jako jsou mobilní telefony, tablety, televize a další. V dnešní době se podpora různých zařízení spíše zužuje na úkor HTML 5, jenž flash nahrazuje.

Výhodou technologie flash, oproti HTML 5 je výpočetní výkon. Je to dáno hlavně programovacím jazykem a jeho kompilací do mezikódu. Je tak možné provádět různé optimalizace, jenž jsou v JavaScriptu jen stěží proveditelné (jde hlavně o detekci datových tipů).

V dnešní době se flash na webu používá již stále méně (právě na úkor HTML 5) a dá se očekávat, že v budoucnu vymizí úplně. To ovšem neznamená, že by tato technologie vymizela. Již nyní lze pomocí Adobe Flash vyvíjet nativní aplikace pro Android, iOS, Windows a Linux.

2.1.2 Microsoft Silverlight

Microsoft Silverlight je poměrně mladá technologie, založená na platformně .NET. Stejně jako Adobe Flash jde o technologii využívající zásuvný modul. Na rozdíl od Adobe Flash ovšem má daleko nižší tržní podíl (zhruba poloviční).

Pro vývoj aplikací je možné využít libovolný textový editor a zdrojové kódy následně zkompilovat pomocí Silverlight SDK. Pro vývoj aplikací je možné využít celou škálu programovacích jazyků, jako je C#, Visual Basic, C++, Python a spoustu dalších.

V dnešní době je tato technologie již v podstatě mrtvá a to z několika důvodů. Microsoft ji již neplánuje dále vyvíjet, Internet Explorer na Windows 8 neumožňuje běh plugin technologií a k dnešnímu dni tuto technologii nepodporuje žádná mobilní platforma.

Hlavní výhodou této technologie oproti HTML 5 je její výpočetní výkon. Je založena na architektuře .NET, kde je kód kompilován do mezikódu a ten následně interpretován. Tento interpret ovšem implementuje just in time kompilaci díky níž se mnohdy dokáže vyrovnat i jazykům kompilovaným do binárního kódu.

2.1.3 JavaFX

Java je technologie společnosti Oracle (dříve vyvíjená společností Sub Microsistems). Stejně jako všechny doposud uvedené technologie využívá pro svůj běh v prohlížeči zásuvný modul. Na rozdíl od ostatních se ovšem těší nejmenší přízni uživatelů i vývojářů (pro použití na webu). Je to dáno hlavně pomalým spouštěním, ale také velmi častými bezpečnostními chybami.

Pro vývoj webových aplikací postavených na technologii Java je určena tzv. JavaFX. Jde v o RIA framework, jenž má vývoj aplikací usnadňovat.

Tato technologie nebyla nikdy široce rozšířena a s příchodem mobilních zařízení zcela ztrácí smysl.

2.1.4 HTML 5

Standard HTML 5 je dílem standardizační organizace **World Wide Web Consortium** a pracovní skupiny **Web Hypertext Application Technology Working Group**. Nejde tedy o produkt jedné jediné společnosti, ale mnoha společností a dokonce mnoha jedinců, protože do těchto skupin může svými nápady, či připomínkami přispívat teoreticky každý. Mezi ty nejvýznamnější přispěvatele patří aktuálně Google, Microsoft, Apple, Mozilla, Opera, Adobe a spousta dalších dalších. Ačkoliv se může zdát, že tento fakt není důležitý, tak opak je pravdou. Nejtěžší totiž není vytvořit nějaký standard. Mnohem těžší je udělat jej skutečně kvalitně, tak aby pokrýval pokud možno vše co se může vývojářům hodit a hlavně tento standard prosadit.

V podstatě jsou dvě možnosti, jak jakoukoliv novou technologii na webu prosadit. Jednou z možností je využití **technologie zásuvných modulů**, jako je tomu u všech výše zmíněných technologií. S touto možností jde, ruku v ruce, spousta problémů. Díky obrovskému množství různých zařízení na dnešním trhu. A to ať už jde o mobilní telefony, tablety, notebooky, počítače, televizory, nebo jiné zařízení a zároveň různorodosti operačních systémů, které na nich běží, jako jsou Windows, Mac OS, Mint, Fedora, Ubuntu, iOS, WebOS, Symbian, Android a spousta další většinou založených na linuxu. Je potřeba pro všechny tyto kombinace vytvořit zvláštní zásuvné moduly a to ještě navíc pro každý z mnoha různých prohlížečů. To je ovšem jen začátek. Druhý problém, který se naskýtá, je jak dostat tento modul k uživatelům. Většina uživatelů si chce nainstalovat prohlížeč (spousta není schopna ani toho) a používat jej. Nechtějí nic, pro ně složitě, instalovat. Těchto problémů existuje ještě více a dostaneme se k nim v textu později. Ovšem již nyní je možné zpětně se zamyslet nad všemi již zmíněnými technologiemi. Žádná z nich není dostupná na všech vyjmenovaných zařízeních, operačních systémech, prohlížečích a jejich kombinacích. Tento fakt je velmi závažný, protože význam těchto zařízení stále roste. Jedním z hlavních důvodů proč tomu tak je, je právě to, že jde většinou o uzavřené technologie jednoho výrobce, který jednak není schopen pokrýt všechny tyto kombinace, a hlavně není schopen se domluvit se všemi výrobci, aby jejich zásuvný modul podporovali.

Tímto jsme se vrátili nazpět k HTML 5, které je v mnohém opakem první z možností. Již bylo uvedeno, že standard HTML 5 je tvořen velmi otevřeným způsobem, kdy mezi sebou lidé a hlavně společnosti komunikují a každá z nich může přispívat svými návrhy a připomínkami. Další důležitou vlastností HTML 5 je, že nejde o implementaci této technologie, ale pouze o její standard. To je velký rozdíl oproti již zmíněným technologiím. Není totiž potřeba přesvědčovat výrobce zařízení aby podporovali ten, či onen zásuvný modul. Jediné, co musí udělat, je nainstalovat do zařízení internetový prohlížeč, což je dnes již téměř samozřejmé. A tento prohlížeč musí podporovat právě HTML 5. I když se nyní může zdát, že je situace stejná, jen místo nutnosti instalace modulu je potřeba pod-

pora nějaké technologie v prohlížeči, tak tomu tak není. Stačí si uvědomit, že internetových prohlížečů, respektive jejich jader, neexistuje zase takové množství. Aktuálně to jsou WebKit, Trident, Gecko, Presto a několik dalších, jejichž podíl je víceméně bezvýznamný. To znamená, že pokud některý z výrobců nainstaluje do svého zařízení prohlížeč, bude obsahovat právě jedno z nich. Ve výsledku je tedy nutné pouze to, aby HTML 5 standard přijaly firmy, které za těmito jádry stojí. Což jsou právě Apple Inc., Google Inc., Microsoft Corporation, Mozilla Foundation a Opera Software, které do standardu HTML 5 samy přispívají.

Jednou z největších výhod využití HTML 5 oproti jiným RIA technologiím je tedy jeho otevřenost a platformní nezávislost. Díky tomu je možné spouštět aplikace v něm napsané téměř na jakémkoliv zařízení s internetovým prohlížečem.

S využitím HTML 5 na různých zařízeních, obzvláště těch mobilních, souvisí další důležitá vlastnost této technologie. A sice šetrnost k systémovým prostředkům a baterii. Toto je mimo jiné jeden z důvodů, proč Apple neumožnila instalaci Flash Playeru na iOS zařízení, nebo proč se snaží Microsoft ve Windows 8 zakázat jakékoliv zásuvné moduly. Ty jsou totiž k systémovým prostředkům velmi nešetrné, což má za následek rychlé vybíjení baterií mobilních zařízení. Velmi častou situací, kdy se s tímto uživatelé setkávají, je při přehrávání videa na internetu. Před příchodem HTML 5 byla jediná možnost jak přehrávat video na webu právě pomocí zásuvných modulů. Jejich základní problém je už sama o sobě podstata jejich fungování. Každá instance modulu si zabírá své vlastní systémové prostředky, které potřebuje pro svůj vlastní běh. A následně ještě systémové prostředky potřebné pro přehrávání videa. Navíc tímto spojením technologií dochází k značné nehomogennosti celé aplikace. Musí se řešit komunikace mezi moduly a stránkou, což je poměrně komplikované a neefektivní. HTML5 proto přichází s možností přehrávat video a audio přímo v prohlížeči, bez potřeby jakéhokoliv zásuvného modulu. To umožňuje mnohem **lépe využívat systémové zdroje**.

Jediné výhody HTML 5 neplynou pouze z nezávislosti na platformách a výkonu. Velmi důležitou vlastností a to hlavně pro budoucnost celého webu je jeho **sémantika**. Ta byla v podstatě hlavním důvodem samotného vzniku HTML. Bylo potřeba nějak říct, že nějaká část textu je nadpis, odkaz na jiný dokument, tučný text, atd. Toto rozlišení je důležité jednak z pohledu uživatele, aby byl schopen se v textu orientovat a správně jej chápat, ale také z pohledu vyhledávání. O významu vyhledávačů se jistě nedá diskutovat. Všichni je každodenně používáme a jsme závislí na jejich schopnosti nalézt právě to co hledáme. Toto je ovšem velmi těžké, až nemožné, bez toho aby vyhledávače byly schopny porozumět obsahu jednotlivých webů. A právě v tomto HTML vysoce převyšuje všechny již zmíněné technologie. Jde o textový formát, takže je velmi jednoduché jej strojově zpracovávat. Navíc HTML, právě od verze 5, obsahuje mnoho nových nástrojů pro popis samotného dokumentu tak aby bylo strojové zpracovávání a porozumění obsahu co možná nejjednodušší. Naproti tomu ostatní RIA technologie jsou distribuovány jako binární soubor, který je velmi obtížné zpracovávat a to i přes to, že se to snaží tvůrci technologií usnadňovat. A také neobsahují nástroje k popisu významu jejich obsahu.

Výše zmíněná sémantika souvisí s další ne méně důležitou vlastností, kterou HTML 5 vysoce převyšuje konkurenci, a tou je **přístupnost**. V naší populaci je velké množství

lidí s nějakým handicapem. Když je řeč o handicapu, nemusí se jednat jen o nějaké postižení nebo poruchu. Handicap může být i druh zařízení, které daný jedinec používá. Příkladem jsou všechna dotyková zařízení, na kterých nelze provádět vše co lze provést s myší. Nebo také rozlišení obrazovky, rychlost internetového připojení a další. Platí zde v podstatě stejné výhody jako byly již uvedeny výše. HTML je textový formát a popisuje sémantiku obsahu. Díky tomuto není problém pro různé specializované nástroje obsah zpracovávat a nabízet uživateli ve formě, kterou je schopen pojmout. Toto technologie založené na zásuvných modulech většinou neumí a pokud ano, tak velmi nedostatečně a to například proto, že vše běží uvnitř zásuvného modulu, jehož obsah není prohlížeč ani jiný specializovaný nástroj schopen pro uživatele jakkoliv optimalizovat.

Jak bylo již uvedeno, HTML 5 není jen značkovací jazyk, ale označuje se tak celá skupina technologií, jako je JavaScript pro aplikační část, CSS pro stylování obsahu, nebo různá API prohlížeče. První dvě zmíněné technologie nejsou novinkou, se kterou by přišlo HTML 5. Byly zde již dříve, ale až s příchodem HTML 5 se dočkaly výrazných změn a hlavně rozšíření o další prvky. Významnou novinkou ovšem jsou **API prohlížeče**.

Jednou z největších výhod již zmíněných RIA technologií, oproti HTML byly mnohé funkce, které jsou potřeba pro vývoj skutečných aplikací. Těmi jsou například databázová uložiště, práce se souborovým systémem, nahrávání souborů na web, zpracovávání binárních dat a mnohá další. S příchodem HTML 5 se ovšem tyto výhody stávají minulostí. A to právě díky API, která poskytuje prohlížeč. Ve skutečnosti šlo HTML 5 i trochu dál a umožňuje některé funkce, které konkurenční technologie neumožňují. Příkladem je geolokace, offline aplikace, nebo možnost notifikace. Blíže se s touto problematikou seznámíme v samostatné sekci.

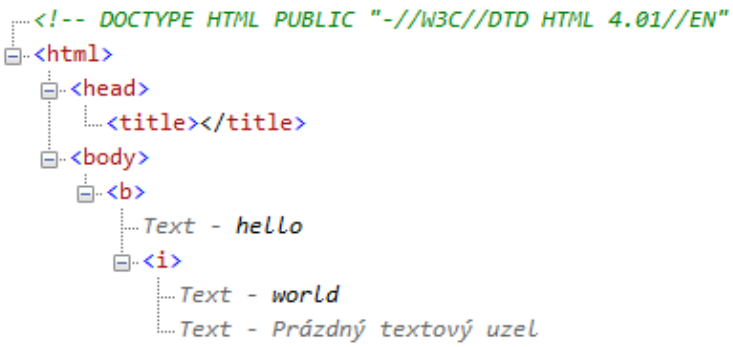
Stejně jako každá jiná technologie má i HTML 5 své nevýhody. V současnosti stojí za zmínku nedostatečná podpora v prohlížečích, která se ovšem stále zlepšuje. A dále pak slabá podpora video a audio formátů. Ta se jeví aktuálně jako největší problém, protože téměř každý prohlížeč podporuje jiný formát, což velmi komplikuje jejich použití na webu. V současnosti jsou to formáty *H.264*, *Ogg Theora* a *VP8*.

Po shrnutí všech výhod a nevýhod jednotlivých technologií vychází, přinejmenším do budoucna, jako vítěz jednoznačně HTML 5. Čemuž naznačuje i to, že firmy stojící za konkurenčními technologiemi se velmi usilovně snaží tyto technologie migrovat právě na HTML 5. Adobe již nyní umožňuje export z flashe do HTML 5. Microsoft naopak podle všeho ukončí vývoj silverlightu jako webové technologie a již nyní se snaží nahrazovat jej právě HTML 5.

2.2 Tagy a sémantika

HyperText Markup Language, jak již název napovídá, je jazyk využívající značek pro tvorbu hypertextových dokumentů. Hypertextový dokument je textový dokument, jenž obsahuje odkazy na další textové dokumenty. Tohoto provázání se dosahuje právě díky značkám. Těmto značkám se říká **tagy**.

Kromě základního tagu pro provázání dokumentů existuje spousta dalších tagů, které slouží k popisu samotného dokumentu. Jelikož se jedná o digitální dokumenty, slouží tento popis ke strojovému zpracování a následné interpretaci uživateli. Aby bylo toto



Obrázek 1: DOM v Internet Exploreru 8

možné, je potřeba jednotlivým tagům definovat **syntaxi** a **sémantiku**. Syntaxe udává jak správně tagy zapisovat, tak aby bylo možné je korektně zpracovávat a sémantika jim dodává jejich význam.

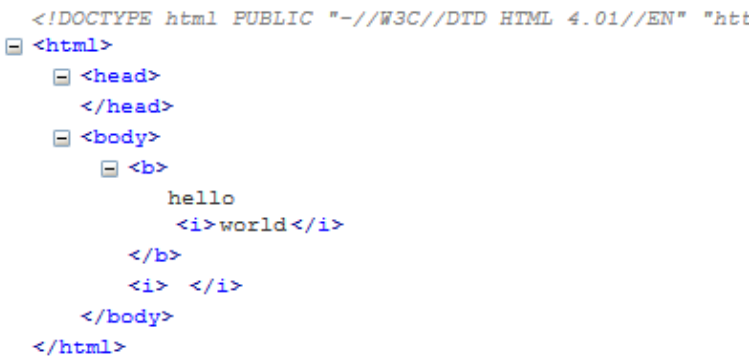
2.2.1 Syntaxe

Syntaxe HTML je na první pohled velmi podobná syntaxi XML, ale není tomu tak. V mnoha případech může být HTML dokument zároveň i XML dokumentem, ale není to zaručeno specifikací, takže s ním není možné tímto způsobem nakládat. HTML dokumenty se tedy zpracovávají speciálními HTML parsery. Velkým problémem ovšem bylo, že až do HTML 5 nebylo nikde definováno, jak se mají tyto parsery chovat a hlavně jak nakládat s chybami v dokumentu. Toto vedlo k tomu, že různé prohlížeče generovaly na totožný HTML dokument odlišné výstupy. Příklad kódu, který generuje ne-jednoznačný DOM je vidět na výpise 1. Na obrázku 1 je vidět výsledný DOM v IE 8 a na obrázku 2, výsledný DOM téhož dokumentu ve Firefoxu. HTML 5 ovšem toto chování řeší a všechny prohlížeče, které jej podporují, tak generují totožný DOM. Toto je důležité hlavně při práci s dokumentem pomocí JavaScriptu.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
  </head>
  <body>
    <b>hello <i>world</b>
  </body>
</html>
```

Výpis 1: Příklad HTML dokumentu s nejednoznačným DOM

To, že nebyl parser v předchozích specifikacích vůbec zahrnut nemá jen negativní stránky. Naopak to vedlo k tomu, že se tvůrci parserů snažili o to, aby se jejich parser



Obrázek 2: DOM ve Firefoxu

dokázal vypořádat i s částečně chybným kódem. Ne, že by to dnes již nedělali, ale právě toto vedlo k tomu, že je HTML 5 zpětně kompatibilní. Prohlížeče, které dokázaly zobrazit stránku napsanou v rámci standardu HTML 4.01, si poradí se stránkou napsanou pod standardem HTML 5. Nové elementy z HTML 5, jenž hrají jen sémantickou úlohu, se vyrenderují, jako kdyby to byly elementy typu <*div*>. Speciální elementy, například pro vložení videa, se nevyrenderují, ale prohlížeč i přesto stránku vyrenderuje. Toto chování je docela důležité, protože u většiny stránek hraje roli právě jejich textový obsah, který uživateli dostupný bude.

HTML 5 přichází s řadou nových tagů. Jejich bližší popis bude v textu uveden později. Ovšem již nyní je potřeba uvést, že co do syntaxe se mnoho nezměnilo. Přibyly pouze nové názvy tagů, ale jejich syntaktický popis je stejný u těch co existují již ze starších verzí. Co se ovšem o proti předchozím verzím HTML změnilo, je daleko větší důraz na sémantiku.

2.2.2 Sémantika

Sémantika je samozřejmě důležitá u všech jazyků, jinak by ani nebyl důvod k jejich existenci, což u HTML platí dvojnásobně. U programovacích jazyků, jako jsou C, C++, C#, Java a další, je sémantika důležitá hlavně pro programátora. Poté co programátor kód dokončí, proběhne kompilace do strojového kódu, který vykonává počítač bez znalosti jakékoliv sémantiky. Jsou to pouze příkazy, které počítač jeden po druhém vykonává bez toho aby potřeboval rozumět jejich kontextu, nebo významu skupin těchto operací. A stejně tak uživatelé, kteří výsledný program spustí, se nezajímají o to co je uvnitř. Nezajímají se o význam jednotlivých operací. Sémantika zde tedy hraje roli hlavně jako nástroj pro vývoj.

U HTML je tomu ale trochu jinak. HTML kód se nekompiluje, ale dostává se k uživateli tak jak je napsán. A až následně jej interpretuje prohlížeč a nějak zobrazí. Nyní se může zdát, že zde sémantika nehraje roli, protože ve výsledku jde k uživateli stejně jen jakýsi „obrázek“ stránky. Je si ale potřeba uvědomit, že jsou mezi námi i uživatelé s určitým handicapem, jimž obraz nestačí. Tito uživatelé často používají speciální čtečky textu,

pro něž je důležitá právě sémantika. Pokud se například označí skupina odkazů jako navigace, čtečka je schopná říct uživateli, že jde o navigaci. Stejně jak hlavička, nadpis, tělo článku a další.

Nyní se může zdát, že je sémantika důležitá pouze pro lidi s nějakým handicapem. Důležité je ale uvědomit si, že to nemusí být jen lidé a nemusí jít jen o čtečky. Mohou to být stroje a na nich běžící software. Pokud navštívíte stránku na které jste ještě nikdy nebyli, nejprve ji podvědomě na základě jejího vzhledu analyzujete. Důležitou roli v tomto hrají i vaše očekávání. Například, že hlavní obsah bude uprostřed, menu nahoře. Větším písmem budou nadpisy, menším pak hlavní obsah. Takováto analýza je pro strojovou analýzu velmi náročná a také velmi nepřesná. Jak bylo uvedeno, u lidí hraje velkou roli jejich očekávání. Pokud je stránka nesplňuje, mají problémy. Tytéž problémy mají stroje. Nejdůležitějšími stroji z pohledu webu jsou vyhledávače. Pokud **vyhledávač** dokáže porozumět obsahu stránky, dokáže ji i lépe zpracovat a na základě toho pak nabízet uživatelům lepší výsledky. Příkladem jedné z mnoha věcí, které v tomto směru HTML 5 nabízí, jsou microdata, kterým bude následně věnována samostatná kapitola. Tato metadata například umožňují vložit informaci o tom, že to, co byl doposud jen nějak naformátovaný text, je akcí, která má nějaký začátek, konec, místo konání a další. Při jakémkoliv strojovém zpracovávání to tedy již není jen text, ale je to plnohodnotná informace, se kterou se dá dále pracovat.

2.2.3 Nové elementy

S každou novou verzí HTML přicházejí nové elementy (tagy). A stejně tak se některé ruší. V textu níže jsou uvedeny všechny nové tagy, které přináší HTML 5. Při jejich popisu je kladen důraz na pochopení jejich sémantiky. Ta je v mnohých ohledech složitější než u dosavadních elementů, což vede k častým chybám v jejich použití.

hgroup je element, který má zastřešovat nadpis úrovně h1 až h6. Tyto nadpisy jsou následně považovány za jeden celek. Na výpise 2 vidíte příklad použití hgroup společně s nadpisy úrovně h1 a h2, kde nadpis h1 hraje úlohu hlavního nadpisu a nadpis h2 roli podnadpisu. Rozdíl mezi případem, kdy se hgroup použije a kdy se nepoužije, je v tom, že jeho použitím říkáme, že jde o jeden jediný nadpis složený z několika částí s různou důležitostí. Teda v ukázce jde o nadpis a podnadpis. Kdežto bez jeho použití by šlo o dva nadpisy. Tento element tedy neříká nic o tom, jak daný obsah vykreslit, ale hraje pouze sémantickou úlohu.

```
<hgroup>
  <h1>ASP.NET</h1>
  <h2>Design Patterns</h2>
</hgroup>
```

Výpis 2: Příklad užití elementu hgroup

header označuje úvod. Rozšířeným omylem, vycházejícím z podobnosti s elementem head, jenž označuje hlavičku HTML dokumentu, je že, element head má sloužit pouze jako označení úvodu stránky. Není to ovšem pravda. Element head má za úkol označovat úvod v nějaké ucelené části. Tedy nejen celé stránky, ale i jednotlivých článků, sekcí a podobně. S tím souvisí jeho provázání s již zmíněným elementem hgroup, který je často jeho obsahem. Na výpise 3 vidíte příklad jeho použití. Ve výpise je záměrně uveden kromě hgroup, také vyhledávací formulář, který je také korektní součástí elementu header.

```
<header>
  <hgroup>
    <h1>ASP.NET</h1>
    <h2>Design Patterns</h2>
  </hgroup>
  <form action="#" method="post">
    <label for="s-box">Phrase</label><input type="search"
       name="q" id="s-box" />
    <input type="submit" value="Search" />
  </form>
</header>
```

Výpis 3: Příklad užití elementu header

footer je v podstatě opakem elementu header. Zatímco header označoval úvod, tak footer označuje patičku. A opět nejde jen o patičku celého dokumentu. Patička se vždy vztahuje k jejímu nejbližšímu nadřazenému elementu. Na výpise 4 vidíte příklad užití elementu footer v rámci patičky článku. Jak bylo uvedeno, patička se vztahuje k nejbližšímu nadřazenému elementu, respektive jeho obsahu, což je element div.

```
<div>
  <header>
    <h1>ASP.NET</h1>
  </header>
  <p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit.
       Phasellus vitae orci erat. Lorem ipsum dolor sit amet,
       consectetur adipiscing elit.
  </p>
  <footer>
    <p>Published 05/07/2012, by Jan Janousek</p>
  </footer>
</div>
```

Výpis 4: Příklad užití elementu footer

nav slouží k definici navigace na stránce. Není ovšem nutné aby byly všechny navigační bloky uvnitř elementu nav. Ten by měl označovat jen důležité navigační prvky, protože jak již bylo zmíněno, je využíván čtečkami. A při velkém množství navigací by se uživateli velmi těžko na webu orientovalo.

Obsah samotného nav elementu není standardem definován. Pouze je řečeno, že má obsahovat navigační prvky.

section element slouží k definici tématických skupin obsahu. Jedním z problémů při strojovém zpracovávání HTML dokumentu bylo vždy jak poznat, že určité části spolu souvisí. Bude-li na stránce nadpis, odstavec následovaný tabulkou, a následně další odstavec, jak poznáme, že jde o jeden celek? Právě toto řeší element section. Ukázka použití je na výpise 5.

```
<section>
  <h1>ASP.NET</h1>
  <p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit.
       Phasellus vitae orci erat.
  </p>
  <table>
    <tr>
      <th>Name</th><th>Weight</th>
    </tr>
    <tr>
      <td>Susan</td><td>45</td>
    </tr>
  </table>
  <p>
    Phasellus vitae orci erat. Lorem ipsum dolor sit amet,
       consectetur adipiscing elit.
  </p>
</section>
```

Výpis 5: Příklad užití elementu section

article slouží jako samoobsažný prvek dokumentu. To znamená, že jeho obsah by měl být natolik ucelený, aby nebyl závislý na obsahu zbytku dokumentu. Nemusí to tedy být jen článek, tak jak by název elementu napovídal. Ve skutečnosti může obsahovat v podstatě jakoukoliv ucelenou část obsahu, jako mohou být widgety, komentáře na blogu, atd. V mnohém je podobný elementu section a pokud není úplně jasné, zda použít section, nebo article, měl by se použít právě article. Ukázka použití je na výpise 6. Mělo by z něj být patrné, jak jej kombinovat s již uvedenými elementy. Jde o jeden celek, který není nijak závislý na zbytku stránky ze které je vyjmut (nese sebou veškeré informace). Element section je zde využit k rozdělení obsahu na dva. Nejde tedy o jeden článek, ale o dva.

```
<article>
  <header>
    <h1>ASP.NET</h1>
  </header>
  <section>
    <h2>Phasellus vitae</h2>
    <p>
      Lorem ipsum dolor sit amet, consectetur adipiscing
         elit. Phasellus vitae orci erat.
    </p>
    <table>
      <tr>
        <th>Name</th><th>Weight</th>
      </tr>
      <tr>
        <td>Susan</td><td>45</td>
      </tr>
    </table>
  </section>
  <section>
    <h2>Adipiscing</h2>
    <p>
      Consectetur adipiscing elit.
    </p>
    <img src="image.png" alt="Example" />
  </section>
  <footer>
    <p>Published 05/07/2012, by Jan Janousek</p>
  </footer>
</article>
```

Výpis 6: Příklad užití elementu article

aside je jeden z mála elementů, jenž jsou závislé na kontextu. Pokud je umístěn uvnitř elementu article, měl by obsahovat informace související s jeho obsahem, ale zároveň by měly nést informační charakter i bez tohoto obsahu. Příkladem může být výpis kapitol. Pokud je element uveden mimo element article, měl by jeho obsah souviset s celým webem. Příkladem můžou být RSS zdroje, sociální widgety, doplňkové navigace a podobně.

Obsah tohoto elementu bývá nejčastěji stylován jako postranní panel.

bdi slouží k vyčlenění části textu, jenž má opačný směr, než okolní text. Texty v angličtině či češtině jsou standardně psány zleva doprava. Naproti tomu texty v arabštině jsou psány zprava doleva.

ruby, rt a rp jsou tři elementy, jenž umožňují používat tzv. Ruby anotace. Ty jsou užitečné pro středoasijské jazyky. V ukázce, kterou můžete vidět na výpise 7, je příklad toho, jak tyto atributy použít. Není zde použito středoasijských znaků, protože jsou pro většinu z nás nečitelné, ale anglických. Přímo do elementu *ruby*, se zadávají tzv. ruby znaky. V našem případě je uvedeno slovo "time". Následně je možné použít nepovinný element *rp*, jehož obsah se zobrazí jen v prohlížečích, jenž *ruby* nepodporují. V našem případě počáteční a koncové závorky. A nakonec se uvede uvnitř elementu *rt* výslovnost těchto znaků. Výsledkem tohoto zápisu je v prohlížečích, jenž ruby podporují, text uvnitř ruby a nad ním menším písmem jeho výslovnost. Prohlížeče, které ruby nepodporují, vypíší text, následovaný závorkami a uvnitř nich výslovnost.

```
<ruby>
  time
  <rp>(</rp>
  <rt>t[ai]me</rt>
  <rp>)</rp>
</ruby>
```

Výpis 7: Příklad užití elementů ruby, rt a rp

mark slouží k zvýraznění části textu. Výsledné označení je velmi podobné tomu, když se inline elementu nastaví CSS vlastnost background-color.

progress slouží k zobrazení ukazatele postupu (progress bar). Jeho použití se předpokládá ve spojení s JavaScriptem, jenž může nastavovat maximální hodnotu a aktuální hodnotu postupu.

meter se může zdát na první pohled stejný jako *progress*. Hlavní rozdíl je v účelu použití. Zatím co *progress* slouží k zobrazení nějakého postupu, meter slouží k vizualizaci skalární hodnoty v rámci nějakého rozsahu. Opět se očekává použití ve spojení s JavaScriptem, ale tentokrát se zadávají tři hodnoty. Minimální hodnotu, maximální hodnotu a aktuální hodnotu. Ukazatel následně zobrazuje relativní hodnotu mezi minimem a maximem (pokus bude minimum 10, maximum 20 a hodnota 15, bude ukazatel zaplněn z 50%).

time slouží, jak název napovídá, k definici času, nebo data. Jde čistě o sémantickou značku, která usnadňuje strojové zpracovávání dokumentu.

wbr umožňuje definovat, kde se mohou v rámci slova zalamovat řádky. Ukázka použití je na výpise 8. Pokud by v textu nebyl tag *wbr* uveden, nemohlo by dojít k zalomení textu v rámci slova. Díky tomu,že uveden je, může dojít k zalomení v místech, kde je uveden.

```
XML<wbr />Http<wbr />Request
```

Výpis 8: Příklad užití elementu wbr

figure a figcaption slouží k vyobrazení obsahu jako jsou obrázky, video, kusy kódu a podobně. A k nim příslušnému popisku. V jistém směru jsou tedy podobné jako element *section*, neboť element *figure*, také může obsahovat více objektů. Ukázka použití je na výpise 9. Popiska je automaticky vykreslena pod obsah k němuž patří.

```
<figure>
  <img src="photo.jpg" alt="Elephant" />
  <figcaption>
    <b>This is really big animal</b>
  </figcaption>
</figure>
```

Výpis 9: Příklad užití elementu figure a figcapture

details a summary řeší častý požadavek na možnost zobrazení jen části textu, jenž si uživatel může následně kliknutím zvětšit. K řešení této funkčnosti byl vždy používán JavaScript. HTML 5 tento problém ovšem řeší za nás, a to pomocí elementů *details* a *summary*. Příklad použití je na výpise 10. Tento kód se postará o to, že se uživateli zobrazí pouze nadpis zkrácený text „Lorem ipsum“ a vedle něj šipka. V okamžiku, kdy na něj uživatel klikne, zobrazí se zbývající text, který je obsahem elementu *details*.

```
<details>
  <summary>Lorem ipsum</summary>
  <p>
    Lorem ipsum dolor sit amet, consectetur adipiscing elit.
       Sed et tellus in nisl consectetur iaculis.
  </p>
  <p>
    Phasellus ullamcorper, massa sed sodales interdum, sem
       tortor fringilla elit,
  </p>
</details>
```

Výpis 10: Příklad užití elementu details a summary

embed slouží k integraci externích aplikací do stránky. Jde tedy o tzv. pluginy, jako je Flash, Silverlight, Java a další. Na tomto elementu je zajímavé to, že ačkoliv byl standardizován teprve v rámci HTML 5, byl používán již dlouho před ním, a to díky jeho podpoře v prohlížečích založených na jádře Gecko (které naopak měly problémy s podporou tehdy již standardního elementu object).

source je element, který může existovat pouze jako součást elementu audio, nebo video. Slouží k popisu zdroje multimediálního obsahu (audia / videa). Pro definici zdroje je potřeba uvést cestu ke zdrojovému souboru, jeho typ a dále je možné uvést, pro jaká média se má použít (projection, screen, tv a další).

Největším problémem audia a videa na webu je v současnosti jeho formát. Specifikace nic neříká o tom, který kodek by měl být použit, což vedlo k tomu, že se každý výrobce prohlížeče snažil použit ten svůj. Výsledný stav je takový, že neexistuje ani jeden audio, nebo video formát, který by byl podporován všemi prohlížeči. V současnosti existují na trhu čtyři nejvýznamnější a těmi jsou WebM, H.264, Ogg Theora a MPEG 4.

track slouží k připojení externích textových stop k audiu, či videu. Tyto stopy mohou mohou být pěti různých typů. Klasické titulky, jenž jsou použity pro překlad dialogů společně s audiem na pozadí. Titulky, jenž jsou použity, pokud není dostupné audio a obsahují nejen dialogy, ale i popisy zvuků. Textový popis děje, jenž se použije, pokud není dostupný obraz. Seznam kapitol, pro navigaci v rámci videa. A posledním typem dat jsou metadata, jenž slouží ke zpracování JavaScriptem.

audio element, jak název napovídá slouží k vložení audia do stránky. To je následně nativně přehráváno. Doposud bylo možné audio přehrávat pouze pomocí zásuvných modulů, které si musel uživatel doinstalovat, což vedlo k velkému množství problémů z nichž ty nejdůležitější byly v rámci tohoto textu již uvedeny.

Na výpise 11 vidíte ukázku použití elementu *audio* ve spojení s elementem *source*. Samotný element audio má několik atributů, které umožňují provést přednastavení. V této ukázce jsou použity atributy „autoplay“, jenž říká, že se má audio začít okamžitě přehrávat, „loop“, který říká, že se má po skončení opět opakovat a posledním je „controls“, jenž udává, že se mají uživateli zobrazit výchozí ovládací prvky.

```
<audio autoplay="true" loop="true" controls="controls">
  <source src="audio.ogg" type="audio/ogg; codecs=vorbis" />
  <source src="song.mp3" type="audio/mpeg" />
  Your browser does not support the audio element.
</audio>
```

Výpis 11: Příklad užití elementu audio

Používat výchozí ovládací prvky ovšem není vždy žádoucí. Obzvlášť proto, že se vzhledově liší prohlížeč od prohlížeče. Díky tomuto se daleko častěji využívá možností JavaScriptu, jenž umožňuje audio element konfigurovat a tím řídit přehrávání videa. Nic tedy nebrání tomu, aby si vývojář vytvořil své vlastní ovládací prvky.

video je dalším z elementů, umožňujících vložení multimediálního obsahu do stránky. Tentokrát videa. Stejně jako element audio se používá ve spojení s elementem *source* a *track*. Hlavní výhodou používání elementu video, oproti použití zásuvných modulů, je lepší optimalizace přehrávání a s tím spojená menší náročnost na systémové zdroje. Není potřeba nic doinstalovávat, neboť potřebné kodeky jsou již součástí prohlížeče, což je důležité obzvláště u mobilních zařízení. Video je skutečně součástí stránky a je na něj tudíž možné provádět veškeré transformace jako je rotace, zkosení, aplikování filtrů a mnoha dalších. A v neposlední řadě lze s videem pracovat pomocí JavaScriptu.

Příklad použití elementu video lze vidět na výpise 12. Element video má opět možnost několika nastavení. Nejdůležitějšími jsou výška a šířka plochy přehrávače. Dále lze například definovat zda se má video automaticky přednačítat i když ještě nebylo spuštěno přehrávání, zástupný obrázek, který se bude zobrazovat dokud se přehrávání nespustí, nebo zda se mají zobrazovat ovládací prvky. Těchto nastavení existuje, stejně jako u audia, více než je zde uvedeno. Všechny nastavení jsou ovšem uvedena v rámci specifikace standardu HTML 5.

```
<video width="960" height="720" preload="true"
   poster="poster.jpg" controls="controls">
  <source src='video.mp4' type='video/mp4;
     codecs="mp4v.20.240, mp4a.40.2"'>
  <source src='video.ogv' type='video/ogg; codecs="theora,
     vorbis"'>
  Your browser does not support the video tag.
</video>
```

Výpis 12: Příklad užití elementu video

canvas je speciální element, který slouží k vytvoření plátna, do nějž je možné kreslit pomocí JavaScriptu. Pro kreslení v současnosti existují dvě základní rozhraní a to 2D a 3D. 2D rozhraní umožňuje kreslit jen základní tvary. Mnohem zajímavější je rozhraní 3D, jenž slouží k renderování hardwarově akcelerované grafiky pomocí WebGL. Příklad použití canvasu je vidět na výpise 13. Tento příklad vykreslí plátno o velikosti 800x600px a pokud není canvas prohlížečem podporován, vypíše hlášku „Your browser does not support the canvas“.

```
<canvas width="800" height="600">
  Your browser does not support the canvas
</canvas>
```

Výpis 13: Příklad užití elementu canvas

Tomu, jak do plátna vytvořeného pomocí elementu canvas kreslit, se bude věnovat kapitola o WebGL.

datalist je speciálním element, který se zásadně liší od všech ostatních. Slouží totiž pouze k definici kolekce dat. Tato kolekce následně může být využívána jinými elementy,

nebo pomocí JavaScriptu. Nejčastější použití je pro vytvoření našeptávače pro input elementy. Ukázka definice kolekce a její přiřazení k input elementu je vidět na výpise 14. V okamžiku, kdy uživatel začne psát název, jenž je v kolekci definován, zobrazí se našeptávač.

```
<datalist id="fruit">
  <option value="Apple">
  <option value="Pear">
  <option value="Cherry">
  <option value="Peach">
  <option value="Banana">
</datalist>
<input type="text" list="fruit" />
```

Výpis 14: Příklad užití elementu datalist

keygen je formulářový prvek, který slouží ke generování privátního a veřejného klíče. V okamžiku odeslání formuláře je privátní klíč uložen do lokálního úložiště klíčů a veřejný je odeslán na server. Očekává se, že by keygen mohl sloužit ke zjednodušení autentizace vůči službám využívajícím TLS a autentizace pomocí certifikátů.

output je formulářový prvek, který slouží k reprezentaci výsledku výpočtu na straně klienta. Jeho obsah není odesílán na server tak jako ostatní formulářové prvky. Jeho význam je čistě sémantický. Neplní tedy žádnou speciální úlohu.

2.2.4 Microdata

Již v části týkající se sémantiky bylo zmíněno, že HTML 5 přichází s něčím co se nazývá „microdata“. Microdata jsou v podstatě typem metadat. Metadata jsou často definována jako „data o datech“. Jde tedy o data, jenž určitým způsobem popisují vlastnosti jiných dat. Samotný pojem „microdata“ by se dal definovat jako **metadata na webu**.

Microdata tedy tedy slouží k popisu dat v rámci webové stránky. Tato data mají následně sloužit pro účely vyhledávačů a prohlížečů, jen jsou schopny tyto data strojově zpracovat a dále s nimi pracovat.

Na výpise 15, je pro lepší pochopení toho, jak microdata fungují, uveden příklad jejich použití.

```
<div itemscope itemtype="http://schema.org/Event">
  <a itemprop="url" href="http://www.vsb.cz/">
    <span itemprop="name">Event name</span>
  </a>
  <p itemprop="description">
    Event description
  </p>
```

```
    <p itemprop="attendee" itemscope
       itemtype="http://schema.org/Person">
      Event organizer: <span
         itemprop="honorificPrefix">Bc.</span> <span
         itemprop="name">Jan Janousek</span>.
    </p>
    <p>
      Event start:
      <meta itemprop="startDate" content="2012-07-08T12:00:00"
         />
      8.7.2012 12:00
    </p>
    <p>
      End date:
      <meta itemprop="endDate" content="2012-07-08T16:00:00" />
      8.7.2012 16:00
    </p>
    <p itemprop="location" itemscope
       itemtype="http://schema.org/Place">
      Location:
      <span itemprop="name">VSB-Technical University of
         Ostrava</span> (
      <span itemprop="address" itemscope
         itemtype="http://schema.org/PostalAddress">
        <span itemprop="streetAddress">
          17.listopadu 15/2172
        </span>,
        <span itemprop="addressLocality">Ostrava-Poruba</span>,
        <span itemprop="postalCode">708 33</span>
      </span>
      )
     </div>
    </p>
  </div>
```

Výpis 15: Příklad užití microdat

Základem definice microdat jsou atributy „itemscope“, „itemtype“ a „itemprop“, jenž mohou být aplikovány na všechny HTML elementy.

Atribut **itemscope** nemá žádnou hodnotu a určuje, že všechen obsah elementu, u nějž je definován, je jedna informace. Tento atribut by nám ale sám o sobě nestačil. Je při nejmenším potřeba definovat, o jaký typ informace se jedná. K tomuto slouží atribut itemtype.

Hodnota atributu **itemtype** musí být absolutní odkaz, nebo více odkazů oddělených mezerou, jenž vedou na slovník popisující daný typ dat. Nikde není definováno odkud

by měly tyto slovníky pocházet, takže si může teoreticky každý vytvořit vlastní. Je si ovšem potřeba uvědomit, že pokud by si každý definoval vlastní slovníky, ztratily by microdata svůj význam. Vyhledávač, nebo prohlížeč totiž musí tyto slovníky znát a být schopný je používat. Za tímto účelem vznikl web **http://www.schema.org/**, jenž obsahuje kolekci celosvětově uznávaných slovníků, které může kdokoliv bezplatně používat. Důležitým faktem je, že se na těchto slovnících shodly nejvýznamnější světové vyhledávače, kterými jsou Bing, Google, Yahoo! and Yandex.

V ukázce z výpisu 15 jsou použity celkem čtyři slovníky, a to pro popis událostí, osob, míst a adres. Jednotlivé slovníky tedy lze kombinovat a vytvářet tak složitější popisy.

Třetím z atributů pro popis microdat je **itemprop**. Jeho hodnota je název vlastnosti, tak jak je uvedena v daném slovníku.

Všechny tři atributy je možné aplikovat na všechny HTML elementy.

Při popisu dat občas nastane situace, kdy by bylo dobré mít možnost vložit data, která uživatel neuvidí, nebo uvidí, ale v jiném formátu, než který je vhodný pro strojové zpracování. Například jako obrázek, nebo v rámci nějakého zásuvného modulu.

V takových případech je možné použít standardní element „**meta**“, jenž je běžně používán v rámci hlavičky stránky. A použít jej v rámci definice microdat. Ukázka použití je k vidění opět na výpise 15. Je zde použit k definici data začátku a konce akce, a to i přesto, že je datum přímo v textu. Důvodem je formát data, které není v rámci textu v univerzálním formátu, který by byl jednoduše zpracovatelný.

Meta element není omezen jen na definici data, ale je pomocí něj možné definovat jakákoliv microdata.

Ačkoliv se může jevit používání microdat poněkud složité, mají tato data svůj přínos a to nejen pro budoucnost, ale již dnes. Jak bylo řečeno, nejdůležitější vyhledávače se již shodly na potřebných slovnících a přinejmenším Google tato data již začal využívat a zobrazovat je ve výsledcích vyhledávání. A to v uživatelsky velmi přívětivé formě, což daný web samo o sobě zvýhodňuje již ve výpise výsledků. Do budoucna se tedy určitě dá očekávat nárůst významu těchto microdat. A to zejména proto, že jediná rozumná cesta, kam dále směřovat vývoj vyhledávačů, je k pochopení samotného obsahu webu.

2.2.5 Data atributy

Při vývoji webových aplikací je velmi často potřeba ukládat data v rámci HTML elementů. Příkladem mohou být nejrůznější JavaScriptové komponenty, jenž se aplikují na HTML kód pomocí doplnění hodnot do atributů *class*, nebo *rel*. Z těch známých bych jmenoval Lightbox, jenž využívá *rel="lightbox"*. Nebo jQuery UI jenž využívá atributy *class* a *id*. Tento přístup má několik problémů. Hlavním je, že tyto atributy k tomuto účelu nejsou určeny. Sémanticky je tedy tento postup chybný. Druhým závažným problémem, jenž vyplývá z toho prvního, jsou samotné hodnoty těchto atributů. Ty jsou značně omezeny a není možné v nich ukládat běžný text s mezerami, diakritikou a dalšími speciálními symboly. Třetím problémem je počet těchto atributů. Dost často potřebujeme uložit více než jednu hodnotu. Přičemž každou hodnotu můžeme uložit jen do jednoho z atributů.

Všechny tyto problémy řeší nové atributy **data-***. Jde o všechny názvy atributů, jenž mají prefix „data-“ a jsou zároveň kompatibilní s XML. Tyto atributy jsou přímo určeny

pro ukládání vlastních dat a jejich následné zpracovávání pomocí JavaScriptu. Z pohledu sémantiky nemají tyto atributy žádný vedlejší účel a neměly by tedy být zpracovávány vyhledávači, ani jiným softwarem, který nepochází přímo od tvůrce daného webu.

Příklad použití data atributů lze vidět na výpise 16. Tento příklad demonstruje využití data atributů ke konfiguraci JavaScriptových komponent, což je zároveň jeden z nejčastějších způsobů využití. V tomto případě jde o konfiguraci modulu fotogalerie. Pro její funkčnost je ještě potřeba JavaScriptový kód, který se o vytvoření galerie postará, ale již není v rámci JavaScriptu potřeba jakákoliv konfigurace. Což usnadňuje používání, klade nižší nároky na HTML kodéra, vede k čistějšímu kódu, snazší udržovatelnosti a navíc je mnohem snazší definovat konfiguraci strojově v rámci generování HTML kódu na serveru.

```
<div data-module="photo-gallery" data-loop="true"
   data-modal="true" data-effect="elastic">
   <img src="photo-1-thumb.jpg" alt="Elephant"
      data-full-size="photo-1.jpg" data-desc="African
      Elephant" />
   <img src="photo-2-thumb.jpg" alt="Monkey"
      data-full-size="photo-2.jpg" data-desc="Monkeys in the
      jungle" />
   <img src="photo-3-thumb.jpg" alt="Giraffe"
      data-full-size="photo-3.jpg" data-desc="Giraffe grazing
      with a herd of zebras" />
</div>
```

Výpis 16: Příklad užití atributů data-*

Jakým způsobem s data atributy v rámci JavaScriptu pracovat, bude popsáno v samostatné kapitole v části o JavaScriptu.

2.2.6 Formuláře

Jak již bylo zmíněno, HTML 5 se spíše než na tvorbu webových stránek, jenž jsou již dostatečně pokryty v předchozích verzích standardu, zaměřuje na vývoj webových aplikací. Nedílnou součástí snad každé aplikace, ať už desktopové, nebo té webové, jsou formuláře. Ty již samozřejmě existovaly v předchozích verzích standardu, ale v rámci HTML 5 prošly zásadním vylepšením, a to hlavně co se týče jejich validace.

Základem pro formuláře ve všech verzích HTML je element „<form>“. V HTML 5 se v tomto ohledu nic nemění, pouze je rozšířen o další dva atributy. „Autocomplete“, jenž říká, zda může prohlížeč automaticky vyplnit formulář za uživatele, a druhým je „novalidate“, pomocí nějž je možné vypnout validaci formuláře.

Daleko více změn než element „<form>“ prodělal element „<input>“, respektive jeho atributy. Doposud bylo možné nastavit typ tohoto formulářového elementu na šest hodnot a to „text“, „password“, „checkbox“, „radio“, „file“ a „submit“. HTML 5 tento seznam rozšiřuje o dalších 13 nových typů. Co je ovšem na těchto typech nejdůležitější je,

že automaticky podléhají **validaci na úrovni prohlížeče**. Není tedy potřeba aby vývojář programoval vlastní JavaScript-ovou validaci, která je navíc dost často chybná, protože napsat korektní regulární výraz například na kontrolu emailové adresy není úplně primitivní.

Nové formulářové prvky nejsou výhodné jen pro vývojáře, ale také pro uživatele, neboť mu zjednodušují zadávání dat. Seznam jednotlivých typů je uveden níže společně s jejich základní charakteristikou.

- **tel** - pole pro telefonní číslo. Nemá automatickou validaci, protože jsou formáty telefonních čísel velmi různorodé.
- **search** - vyhledávací pole. Má hlavně sémantický význam. Většina prohlížečů k němu automaticky doplňuje křížek pro rychlé vymazání obsahu.
- **url** - automaticky validovaná absolutní URL adresa.
- **email** - pole pro zadání emailové adresy, jenž je automaticky validováno.
- **datetime** - zadání data a času v UTC. Datum a čas je automaticky validován a uživateli je pro jednoduché zadání hodnot automaticky zobrazen kalendář.
- **date** - zadání data. Data jsou automaticky validována a uživateli je pro jednoduché zadání automaticky zobrazen kalendář.
- **month** - formulářový prvek pro zadání měsíce a roku. Datum je automaticky validováno a uživateli je pro jednoduché zadání automaticky zobrazen kalendář.
- **week** - zadání týdne v roce. Data jsou automaticky validována a uživateli je pro jednoduché zadání automaticky zobrazen kalendář.
- **time** - zadání času. Čas je automaticky validován a uživateli je pro jednoduché zadání zobrazen speciální ovládací prvek.
- **datetime-local** - zadání data a času bez časové zóny. Je naprosto stejný jako „datetime“, jen se očekává znalost časového pásma uživatele.
- **number** - desetinné číslo. Hodnota je automaticky validována a pro snazší zadávání je zobrazeno krokování. Input elementu s tímto typem lze dále přiřadit atributy min, max pro zadání minimální a maximální hodnoty. A atribut step jimž omezuje vstupní data jen na hodnoty minimální získané jako minimální hodnota + n-násobek hodnoty „step“, kde „n“ je celé číslo.
- **range** - číslo z definovaného intervalu. Hodnota je automaticky validována a pro snazší zadávání je zobrazen posuvník. Input elementu s tímto typem lze dále přiřadit atributy min, max pro zadání minimální a maximální hodnota. A atribut step, jimž omezuje vstupní data jen na hodnoty získané jako minimální hodnota + n-násobek hodnoty „step“, kde „n“ je celé číslo.

- **color** - barva v hexadecimální podobě. Hodnota je automaticky validována a pro snazší zadávání je zobrazena paleta barev, ze které lze vybrat.

Třetí skupinou novinek týkajících se formulářů v HTML 5 jsou nové atributy aplikovatelné téměř na všechny dosavadní i nové formulářové prvky. Těmito atributy jsou:

- **autofocus** - při načtení stránky se automaticky nastaví focus na formulářový prvek s tímto atributem.
- **placeholder** - slouží jako popisek obsahu formulářového pole. Zobrazí se uvnitř pole ve formě zašedlého textu a v okamžiku, kdy dostane daný prvek focus, dojde ke skrytí tohoto textu.
- **required** - je základním validačním pravidlem, jenž říká, že uživatel musí do formulářového prvku, jemuž je tento atribut přiřazen, zadat nějakou hodnotu.
- **pattern** - slouží k validaci obsahu formulářového prvku na základě regulárního výrazu, jenž se zadá jako hodnota atributu.
- **novalidate** - umožňuje vypnout validaci na daném formulářovém prvku.
- **autocomplete** - povolní, nebo zakáže automatické vyplňování formulářového pole prohlížečem.
- **dirname** - definuje, že má být společně s daným formulářovým prvkem odeslána informace o směru textu (zprava doleva, či naopak).
- **multiple** - lze definovat na na input elementu typu „file“. Umožňuje výběr většího počtu souborů na jednou a jejich hromadné nahrání na serveru.
- **form** - tento atribut umožňuje zadat ID formuláře, ke kterému má prvek patřit a to bez nutnosti toho aby byl daný prvek uvede uvnitř daného elementu „form“.
- **formaction** - umožňuje přepsat hodnotu atributu „action“ celého formuláře, pokud je tento formulář odeslán pomocí tlačítka s tímto atributem.
- **formenctype** - umožňuje přepsat hodnotu atributu „enctype“ celého formuláře, pokud je tento formulář odeslán pomocí tlačítka s tímto atributem.
- **formmethod** - - umožňuje přepsat hodnotu atributu „method“ celého formuláře, pokud je tento formulář odeslán pomocí tlačítka s tímto atributem.
- **formnovalidate** - - umožňuje přepsat hodnotu atributu „novalidate“ celého formuláře, pokud je tento formulář odeslán pomocí tlačítka s tímto atributem.
- **formtarget** - - umožňuje přepsat hodnotu atributu „target“ celého formuláře, pokud je tento formulář odeslán pomocí tlačítka s tímto atributem.

3 Návrh a implementace komponent

3.1 WYSIWYG Editor

Nedílnou součástí systémů pro správu obsahu je textový editor. Ty se obecně dělí do třech skupin.

První skupinou jsou „plain text“ editory, kde uživatel skutečně pracuje s textem. Jde o uživatelsky nejméně přívětivý způsob, protože neumožňuje obsah jakkoliv formátovat. Příkladem jsou standardní HTML formulářové elementy „textarea“ a „input“.

Druhou skupinou jsou tzv. „markup“ editory. Tyto již umožňují obsah formátovat a to pomocí speciálních značkovacích jazyků, jenž jsou následně přeloženy do HTML (některé editory využívají přímo HTML a není tedy nutný překlad). Nevýhodou těchto editorů je nutnost alespoň základní znalosti daného značkovacího jazyka uživatelem. Příkladem takovýchto editorů jsou „markItUp!“, „Texy!“ a další.

Třetí skupinou jsou tzv. „What You See Is What You Get“, neboli „to co vidíš je to co dostaneš“ editory. Jde o kombinaci dvou předchozích, jenž si bere z každé skupiny to nejlepší. Uživatel pracuje přímo s textem, ale zároveň je mu umožněno formátování pomocí HTML, jenž probíhá interně. Díky tomuto jsou tyto editory uživatelsky nejpřívětivější. Nevyžadují totiž žádné speciální znalosti uživatele a ten okamžitě vidí, jak bude obsah na webu vypadat. Příkladem těch nejznámějších a nejpoužívanějších jsou „TinyMCE“, „CKEditor“, „elRTE“ a další.

Zástupců jednotlivých skupin existuje spousta. Nabízí se tedy otázka, jaký přínos má vývoj vlastního editoru. Nedostatkem drtivé většiny existujících WYSIWYG editorů je, že ve skutečnosti nejsou zcela WYSIWYG. Všechny výše zmíněné editory jsou určeny k použití v rámci administrace. Samotná editace probíhá v rámci jakéhosi sandboxu, kdy je editovaný obsah oddělen od obsahu stránky, na níž editace probíhá. Podoba obsahu tedy v okamžiku editace neodpovídá výsledné podobě na stránce, kde bude obsah umístěn.

Tento problém lze ovšem vyřešit. Stačí přesunou editaci obsahu z administrace přímo na stránku, kde bude obsah vložen. A právě tento problém řeší tento nový WYSIWYG editor.

Základním předpokladem pro vznik editoru je podpora HTML atributu „contenteditable“ v prohlížečích. Pokud je tento atribut aplikován na některý z HTML elementů, stává se jeho obsah editovatelným uživatelem. Tento atribut je sice standardizován až v rámci HTML 5, což by mohlo naznačovat že bude málo rozšířen. Ale ve skutečnosti existuje již od doby vzniku prohlížeče Internet Explorer 5.5, kde byl poprvé implementován. A ostatní prohlížeče na sebe nenechaly s implementací čekat.

Samotný „contenteditable“ ovšem umožňuje pouze editaci obsahu a ne jeho stylování, popřípadě práci s objekty jako jsou tabulky, obrázky, video a další multimediální obsah.

Vybraná oblast

Pro další práci s obsahem v rámci editoru je potřeba vytvořit nástroj pro práci s uživatelem definovanou oblastí (pomocí myši, klávesnice apod.). K tomuto slouží JavaScript-ové

objekty „Range“ v moderních prohlížečích a „TextRange“ ve starých. Ty nejsou vzájemně kompatibilní a je potřeba je funkčně sjednotit. Toto sjednocení není zcela jednoduché a to díky značným odlišnostem v tom co objekty nabízejí. Ale sjednocení je nutné, neboť jiný způsob jak získat informaci o uživatelem vybraném obsahu neexistuje.

S tímto souvisí možnost uložení výběru a jeho obnovení. Pokud uživatel vybere nějaký obsah a následně klikne kamkoliv v rámci stránky, dojde k zrušení tohoto výběru. Uživatel ovšem potřebuje klikat na příslušné ovládací prvky, aby obsah naformátoval. Zároveň je vhodné, aby došlo k obnovení výběru v okamžiku, kdy formátování proběhne.

Největším problémem při ukládání a následném obnovování výběru je, že lze uložit pouze pozici začátku a konce výběru a tu také obnovit. Při formátování, které bude teprve popsáno, dochází v obsahu ke změnám, jenž nemusíme být schopni identifikovat. Což vede k tomu, že známe přesnou pozici výběru před formátováním, ale nejsme schopni určit pozici po jeho skončení. Řešením tohoto problému je přidat do editovaného obsahu speciální značky na místo začátku a konce výběru. A po skončení formátování je opět odstranit a nastavit výběr na pozici těchto značek. Na výpise 17 je uveden kód, jenž při zavolání metody „saveRange“ provede uložení výběru pomocí těchto značek a při zavolání metody „restoreRange“, jej obnoví. Tento kód je implementací pro moderní prohlížeče pomocí „Range“. Pomocí „window.getSelection“ se získá objekt reprezentující výběry na stránce. Pomocí jeho metody „getRangeAt“ lze následně získat konkrétní souvislou vybranou oblast na stránce. Tedy objekt typu „Range“. Ten obsahuje vlastnosti přesně popisující výběr. Z nichž nejdůležitější jsou „[start|end]Container“, jenž obsahuje HTML element v němž výběr začíná/končí a „[start|end]Offset“, jenž určuje pozici začátku/konce výběru v rámci těchto HTML elementů.

```
function createMark(id){
  var mark = document.createElement('span');
  mark.setAttribute('id', id);
  return mark;
}

function saveRange(r){
  var randKey = Math.random().toString().replace('.', '');
  var ids = {
    'start': 'rng-start-' + randKey,
    'end': 'rng-end-' + randKey
  };
  var n1 = createMark(ids.end);
  var sN = r.startContainer;
  var sO = r.startOffset;
  r.collapse(false); // Přesuneme začáte výběru na jeho konec
  r.insertNode(n1); // Vloží značku
  var n2 = createMark(ids.start);
  r.setStart(sN, sO); // Nastaví začátek výběru
  r.setEnd(sN, sO); // Nastaví konec výběru
  r.collapse(false); // Přesuneme začáte výběru na jeho konec
  r.insertNode(n2); // Vloží značku
  return ids;
}

function restoreRange(selection, rng){
  var startNode = document.getElementById(rng.start);
  var endNode = document.getElementById(rng.end);
  var r = selection.rangeCount ? selection.getRangeAt(0) :
     document.createRange();
  r.setStartAfter(startNode); // Nastaví začátek výběru za
     počáteční uzel
  r.setEndBefore(endNode); // Nastaví konec výběru před
     koncový uzel
  selection.removeAllRanges(); // Odstraní dosavadní výběry
  selection.addRange(r, true); // Přidá nově vytvořený výběr
  startNode.parentNode.removeChild(startNode); // Odstraní
     značku
  endNode.parentNode.removeChild(endNode); // Odstraní značku
}

var selection = window.getSelection();
var range = selection.getRangeAt(0);
var savedRange = saveRange(range);
restoreRange(selection, savedRange);
```

Výpis 17: Uložení a obnovení výběru

Formátování

Formátování obsahu by mělo standardně probíhat pomocí následujících JavaScript-ových metod:

- **execCommand** - aplikuje požadované formátování na aktuálně vybraný obsah, popřípadě jej zruší, pokud již aplikováno je.
- **queryCommandEnabled** - vrací logickou hodnotu true/false podle toho, zda je možné na aktuální výběr aplikovat dané formátování.
- **queryCommandIndeterm** - vrací logickou hodnotu true/false podle toho, zda je možné zjistit zda je na aktuální výběr dané formátování aplikováno.
- **queryCommandState** - vrací logickou hodnotu true/false podle toho, zda při potencionálním pokusu o aplikování daného formátování, dojde k jeho přidání, nebo odebrání.
- **queryCommandSupported** - vrací logickou hodnotu true/false podle toho, zda je daný typ formátování prohlížečem podporován, či nikoliv.
- **queryCommandValue** - vrací hodnotu spojenou s daným aplikovaným formátováním (například při aplikovaném fontu vrátí název tohoto fontu).

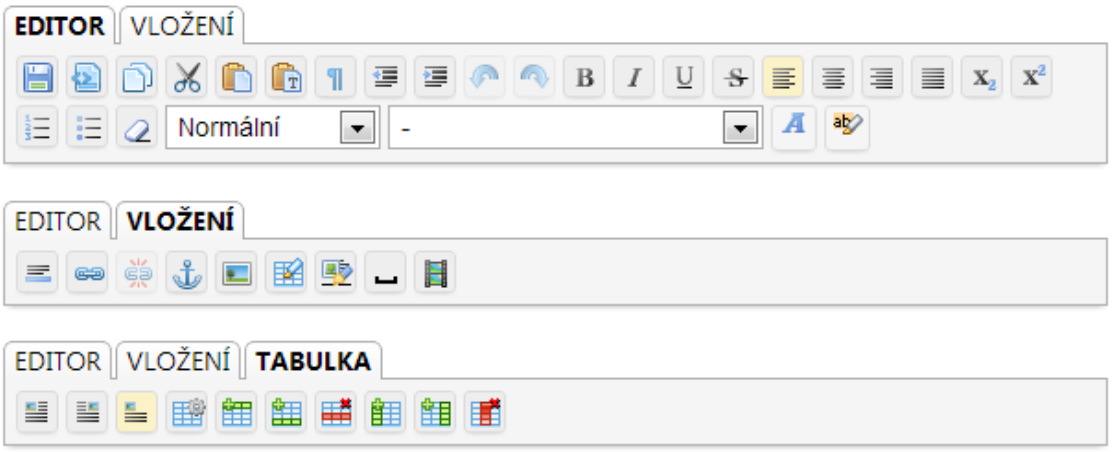
Všechny uvedené metody je potřeba spouštět na objektu „document“. Metoda „execCommand“ má tři parametry. Prvním je identifikátor příkazu, respektive formátování, jenž může nabývat hodnot „bold“, „italic“, „fontname“, „createlink“ a další. Druhý parametr je logická hodnota true/false, jenž určuje, zda se má k tomuto příkazu zobrazit uživatelské rozhraní. Žádný z prohlížečů jej ovšem nepodporuje a hodnota je ignorována. Třetím parametrem je hodnota pro daný příkaz, kterou může být například název fontu.

Všechny ostatní uvedené metody mají pouze jeden parametr a to identifikátor příkazu. Vytvoření editoru, s využitím těchto metod, by mělo být po teoretické stránce velmi jednoduché, neboť veškeré formátování provádí prohlížeč. Opak je ovšem pravdou. Implementace jednotlivých typů formátování je napříč prohlížeči velmi špatná. Prohlížeče jednotlivé formátování buď to vůbec nepodporují, aplikují je odlišně od ostatních prohlížečů, nebo dokonce implementují zcela chybně. Hlavním důvodem k současné špatné podpoře jednotlivých formátování je, že způsob jejich fungování není pevně definován v rámci žádného standardu.

Důsledkem těchto rozdílů je nutnost vlastní implementace většiny typů formátování v rámci JavaScriptu. Tato implementace spočívá v traverzování nad DOM, jenž je velmi pomalé a komplikované, neboť je při aplikování formátování potřeba dodržovat všechna pravidla definovaná standardem HTML.

Editor

Navržené řešení se skládá ze tří částí. Základem je jádro editoru, jenž se stará o běh komponenty, poskytuje rozhraní pro její konfiguraci, obsluhu a propojení zbylých dvou částí.



Obrázek 3: Nástroje WYSIWYG editoru

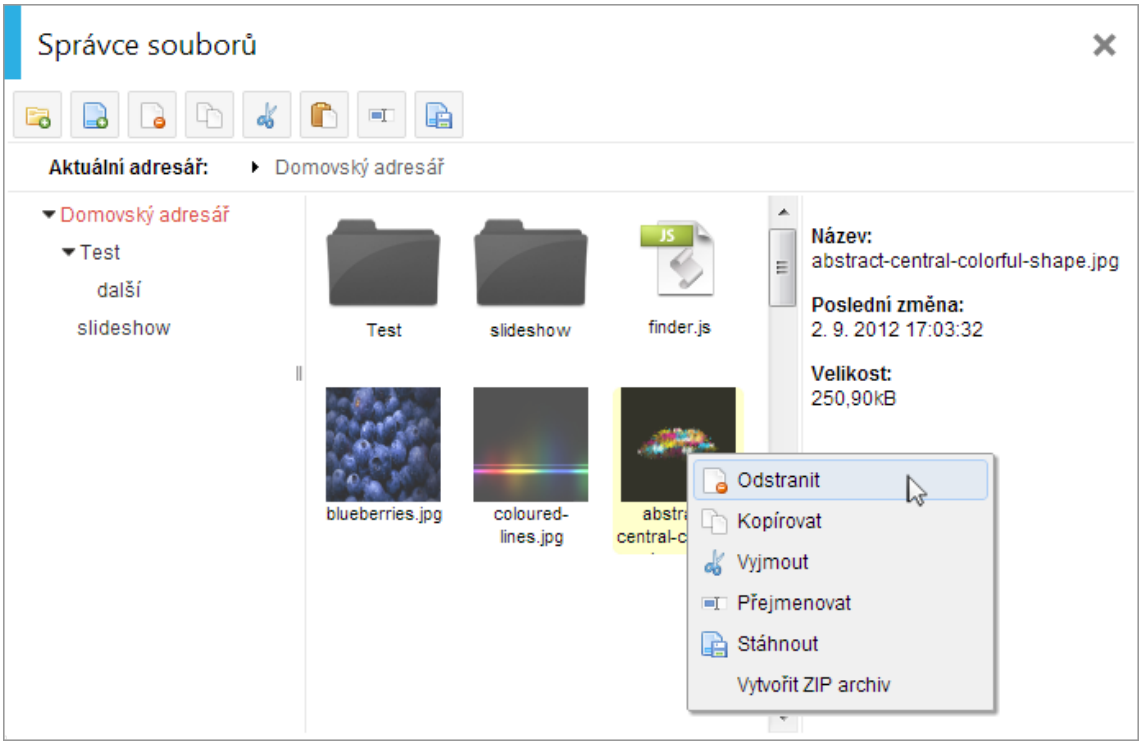
První z nich je uživatelské rozhraní, jenž je navrženo tak, aby bylo nezávislé a bylo jej tedy možné nahradit jiným. Výchozí uživatelské rozhraní je navrženo jako plovoucí ribbon, jehož ukázka je na obrázku 3. Ten se na stránce zobrazí v okamžiku kliknutí do editovatelné oblasti a je jediným uživateli viditelným ovládacím prvkem editoru. Zbytek stránky zůstává nezměněn, díky čemuž je zajištěno, že uživatel po celou dobu editace vidí stránku tak, jak bude vypadat pro běžné návštěvníky. Ribbon je navržen tak, aby se choval dle aktuálního kontextu. Při označení obrázku se zobrazí paletka s nastavením jeho vlastností. Stejně tak pokud je kurzor uvnitř tabulky, popřípadě je označen multimediální obsah jako vide, nebo flash.

Druhou částí jsou nástroje. Ty jsou tvořeny jako moduly. Lze tedy velmi jednoduše konfigurovat, které nástroje budou v rámci editoru dostupné a vytvoření nových modulů spočívá pouze v implementaci patřičného rozhraní. Standardní součástí editoru jsou kromě základních modulů pro formátování textového obsahu moduly pro práci s tabulkami, obrázky, multimediálním obsahem v podobě Adobe Flash, nebo videa. Vkládání odkazů na soubory a obrázků lze navíc usnadnit propojením se správcem souborů, jenž je jednou s komponent, jenž jsou součástí této diplomové práce.

Na výpise 18 je uvedena ukázka použití a základní konfigurace tohoto editoru.

```
$('#id-elementu').contentEditable({
  'finder': function(callback){
    // TODO: Kód pro otevření správce souborů
  },
  'saveCallback': function(html){
    // Tato metoda je zavolána při ukládání obsahu z editoru
  }
});
```

Výpis 18: Příklad použití WYSIWYG editoru



Obrázek 4: Správce souborů

3.2 Správce souborů

Jednou z nejběžnějších činností na počítači je práce se soubory. K tomuto účelu existuje velké množství nástrojů, od příkazové řádky, až po aplikace s vyspělým grafickým rozhraním. Toto ovšem neplatí pro webové aplikace. Nástroje pro správu souborů na webu sice existují, ale svými funkcemi a hlavně uživatelskou přívětivostí silně zaostávají za těmi desktopovými.

Cílem této komponenty je zjednodušit správu souborů na webu a po uživatelské stránce ji co nejvíce přiblížit správě souborů, tak jak je uživatelé znají z desktopu.

Výsledné řešení je možné vidět na obrázku 4. Mezi klíčové vlastnosti správce souborů patří:

- Hromadné nahrávání souborů.
- Nahrávání souborů, či dokonce celých adresářů pomocí „drag & drop“.
- Podpora klávesových zkratek pro kopírování a vložení, výběr souborů, odstranění a dalších.
- Správa souborů a adresářů pomocí „drag & drop“.
- Možnost hromadného výběru souborů.

- Komprese a dekomprese *.zip archivů.
- Možnost změny typu zobrazení a řazení souborů.
- Kontextová nabídka u souborů i adresářů pro intuitivnější práci s nimi.
- Definice oprávnění na soubory a adresáře.
- Rychlé získávání informací o souboru, jako je název, velikost, datum poslední změny, rozměry u obrázků.
- Uživatelské rozhraní známé z desktopových aplikací

Tento text se nebude věnovat podrobnému popisu implementace, neboť většina funkcionality nevyužívá novinky z HTML 5 a neměla by tedy pro tuto práci přínos. Text se bude věnovat pouze implementaci drag & drop souborů a následnému nahrání souborů na server, jenž je novinkou v HTML 5. A dále pak základnímu nástinu implementace serverové části.

Drag & Drop, File API, FileSystem API a XMLHttpRequest 2

Drag & drop, neboli „táhni a pusť“, slouží k přesouvání objektů pomocí myši. Z technického hlediska nejde o žádnou novinku. Již spoustu let existují JavaScript-ové implementace tohoto způsobu ovládání, jenž umožňují přesouvat elementy v rámci webové stránky. V rámci HTML 5 je tento způsob ovládání standardizován a přesunut na nativní úroveň prohlížeče. Tímto je teoreticky zajištěno korektní fungování napříč prohlížeči, ale co je důležitější, nativní drag & drop se neomezuje pouze na přesouvání HTML elementů v rámci HTML stránky. Obsah lze nově přesouvat nejen v rámci jedné HTML stránky, ale také mezi stránkami, či dokonce mezi souborovým systémem počítače a webovou stránkou. A to oboustranně.

V rámci správce souborů je drag & drop využit pro nahrávání souborů na server. Uživatel si v rámci počítače najde adresář, jehož obsah chce nahrát a přesunutím myši jej nahraje na server. A to nejen soubory, ale dokonce celé adresáře. Výsledek je uživatelsky velmi přívětivý a hlavně intuitivní.

Základem pro implementaci jsou následující události:

- **dragstart** - ať už je událost registrována na jakémkoliv HTML elementu, je vyvolána v okamžiku začátku tažení obsahu v rámci prohlížeče.
- **dragenter** - událost je vyvolána, pokud je obsah přesunut nad HTML element, na kterém je registrována.
- **dragover** - událost je volána, pokud je obsah přesunut nad HTML element, na kterém je registrována a to při každém pohybu myši.
- **dragleave** - událost je vyvolána, při přesunutí obsahu mimo HTML element, na kterém je událost registrována.

- **dragend** - ať už je událost registrována na jakémkoliv HTML elementu, je vyvolána v okamžiku ukončení tažení obsahu v rámci prohlížeče.
- **drop** - událost je volána v okamžiku přesunutí obsahu do stránky.

Aby bylo možné přesouvat obsah v rámci stránky, je potřeba HTML elementům, jenž má být možné přesouvat, doplnit atribut „draggable“ s hodnotou „true“. V rámci správce souborů je ovšem potřeba přesouvat pouze soubory z počítače a tohoto atributu tedy není zapotřebí.

Nejdůležitější vlastností, již uvedených událostí, je „dataTransfer“. Ta obsahuje všechny informace spojené s taženým, nebo již přesunutým obsahem. Tato vlastnost je objektem, jenž umožňuje nastavit efekt při přesouvání, zástupný obrázek, jenž bude vidět při tažení obsahu, nastavit, či získat tažená data v textové podobě a hlavně seznam přesouvaných souborů, či adresářů.

Většina dnešních prohlížečů podporuje pouze přesouvání souborů. Ty se z objektu „dataTransfer“ získají pomocí vlastnosti „files“, jako seznam objektů typu „File“. Ty je následně možné dále zpracovávat pomocí tzv. File API. Google Chrome, v současnosti jako jediný, umožňuje také přesouvání adresářů. Ty je možné získat společně se soubory z objektu „dataTransfer“ pomocí vlastnosti „items“, jako seznam objektů typu „FileEntry“ a „DirectoryEntry“, jenž jsou součástí tzv. FileSystem API.

FileSystem API je technologie umožňující přístup JavaScriptu k souborovému systému uživatele, respektive k jeho části. Pro každou webovou stránku využívající FileSystem API je vytvořen v rámci souborového systému sandbox, do něhož je možné ukládat libovolná data a následně je z něj číst. Možnosti práce se soubory v tomto sandboxu jsou srovnatelné s možností práce se soubory v jazycích jako je Java, či C#.

Soubory a adresáře jsou při přetažení přístupny jen pro čtení, což stačí k tomu, aby bylo možné získat informace o adresářové struktuře a souborech, jenž je možné převést na objekty typu „File“ a ty následně zpracovat pomocí File API.

File API slouží k vytvoření a zpracování souborů v paměti. Na rozdíl od FileSystem API tedy neumožňuje zápis na disk. File API mimo jiné umožňuje číst obsah soubor a to jak v binární, tak textové podobě. Pro účely správce souborů ovšem není nutné jeho obsah načítat a je možné jej přímo nahrát na server pomocí XMLHttpRequest 2.

XHR2 je druhá generace XMLHttpRequest, což je technologie, jenž je častěji označována jako AJAX. Tato druhá generace rozšiřuje možnosti této technologie a to mimo jiné, právě o nahrávání souborů. To staví na objektu typu „FormData“, jenž je součástí XHR2 a slouží k vytvoření seznamu obsahujícího dvojici klíč a hodnota. Přičemž jednou z podporovaných hodnot je právě objekt typu „File“.

Ukázku základů použití drag & drop, spojenou s nahráním souboru na sever s pomocí File API a XHR2 je možné vidět na výpise 19.

```
function uploadFiles(files) {
   var fData = new FormData();
   for (var i = 0; i < files.length; i++) {
      fData.append("file_" + i, files[i]);
   }
```

```
    var xhr = new XMLHttpRequest();
    xhr.open('POST', '/upload/');
    xhr.onload = function (e) {
       if (this.status == 200) {
          // TODO: Soubory byly nahrány
       }
    };
    xhr.send(fData);
}
function onDrop(e) {
    e.stopPropagation();
    e.preventDefault();
    uploadFiles(e.dataTransfer.files);
}
var dropArea = document.getElementById('dropArea');
dropArea.addEventListener('drop', onDrop, false);
```

Výpis 19: Nahrání souborů pomocí drag & drop

Správa souborů na serveru

Serverová část správce souborů se skládá z komunikačního rozhraní, jenž je tvořeno jedinou třídou, která mapuje příkazy, jenž přichází od klienta, na příslušné metody a vrací nazpět příslušná data. Názvy příkazů jsou převzaty z operačního systému Linux. Jde tedy o standardní příkazy jako „ls“, „cd“, „mkdir“ a další.

Zmíněné komunikační rozhraní mapuje příkazy na metody rozhraní „IStore“. Přičemž konkrétní implementace je instanci komunikačního rozhraní předána pomocí dependency injection. Díky tomu, že je využito rozhraní a ne třídy, je možné velmi jednoduše zaměnit typ úložiště. Nemusí tedy jít pouze o fyzickou práci se soubory na disku, ale je možné využívat jako uložiště databázi, cloudové služby a další.

Data zasílaná ze serveru ke klientovi jsou ve formátu JSON, kvůli snadnému a rychlému zpracováváni v rámci JavaScriptu.

Příklad použití

Ukázka toho, jak komponentu použít je na výpise 20. Správce souborů je možné použít jednak ve formě dialogu, stejně jako na ukázkovém příkladu. Nebo jej vložit na libovolné místo na stránce pomocí metody „finder(...)“.

V ukázce je použito pouze základní nastavení. Mimo jiné lze dále nastavit maximální velikost nahrávaných souborů, či omezení na výběr adresářů.

Metoda „callback“ v ukázce slouží pro zpracování výběru uživatele. Jako jediný parametr přebírá objekt reprezentující vybraný soubor, či adresář. Ten obsahuje jednak cestu k souboru, velikost, MIME typ, datum poslední změny a pokud je vybrán obrázek, tak jeho rozměry.

```
$('<div>').finderDialog({
  'callback': function (obj) {
    var fileUrl = obj.GetUrl());
    // Callback po vybrání souboru
  },
  'connectorPath': '/finder/', // URL komunikační brány
  'dialogWidth': 800, // Šířka dialogu
  'dialogHeight': 600, // Výška dialogu
});
```

Výpis 20: Příklad použití správce souborů

3.3 JSON formuláře

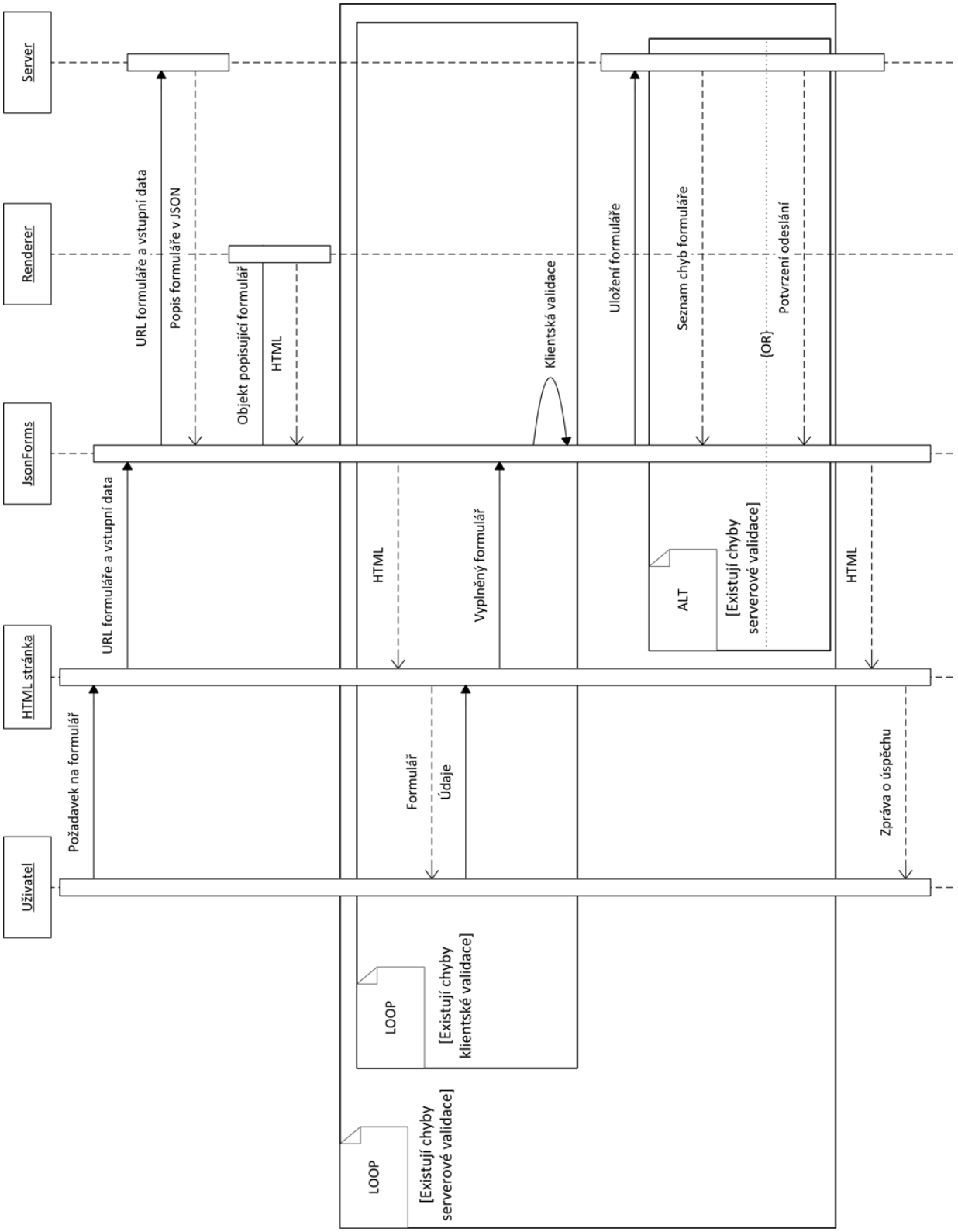
Hlavním rozdílem mezi webovou stránkou a webovou aplikaci je způsob interakce mezi ní a uživatelem. Webové stránky jsou v tomto ohledu jakýmsi statickým prvkem a uživatel je v roli „pozorovatele“. Webové aplikace naproti tomu staví na vzájemné výměně dat mezi uživatelem a samotnou aplikací. Chovají se tedy vůči uživateli dynamicky.

Standardním způsobem jak zajistit vstup dat od uživatele, jsou formuláře. Ty nemají za úkol pouze umožnit uživatelsky přívětivé zadání vstupních dat. Jejich nedílnou součástí je také jejich validace. Vytváření a údržba takovýchto formulářů není zcela jednoduché. Obzvláště ne na webu, kde část aplikace běží u uživatele a část na serveru.

Aby bylo možné práci s formuláři co možná nejvíce zjednodušit, vznikla komponenta s názvem „JsonForms“, která má nahradit již existující nástroj pro práci s formuláři, jenž je standardní součástí ASP.NET MVC. Důvodem k jejímu vzniku je, že stávající formuláře nejsou zcela vhodné pro vývoj webových aplikací založených na HTML 5 a JavaScript-u. Jedním z hlavních nedostatků je renderování na straně serveru, jenž je spojeno s nutností existence šablony, v rámci níž se bude formulář renderovat. Tento přístup je do značné míry omezující, neboť zbytek uživatelského rozhraní tvoří na straně klienta JavaScript. S tímto souvisí obtížná rozšiřitelnost o další formulářové prvky (hlavně ty nestandardní) a nutnost přenosu většího množství dat mezi klientem a serverem. Další z nevýhod je navázání formulářů na entity modelu. Tento přístup je z jednoho úhlu pohledu přínosem, neboť může vývojáři usnadnit práci. Pokud je ovšem potřeba vytvořit formulář jen pro část atributů dané entity, vzniká problém s validací povinných položek této entity. Ten se dá samozřejmě řešit, a to několika způsoby, ale všechny jsou komplikované.

Komponenta se skládá ze dvou částí, a to serverové a klientské. Serverová část se stará pouze o generování popisu formulářových prvků a závěrečnou validaci, poté co byl formulář již validován na straně klienta a následně odeslán na server. Klientská část se stará o komunikaci se serverem pomocí AJAX-u, renderování a klientskou validaci.

Pro komunikaci mezi klientskou a serverovou částí je využit formát JSON, který umožňuje stejně kvalitní popis jako formát XML, ale práce s ním v rámci JavaScript-u je jednodušší a rychlejší. Kompletní životní cyklus formuláře interakce uživatele a validace je znázorněn na obrázku 5.



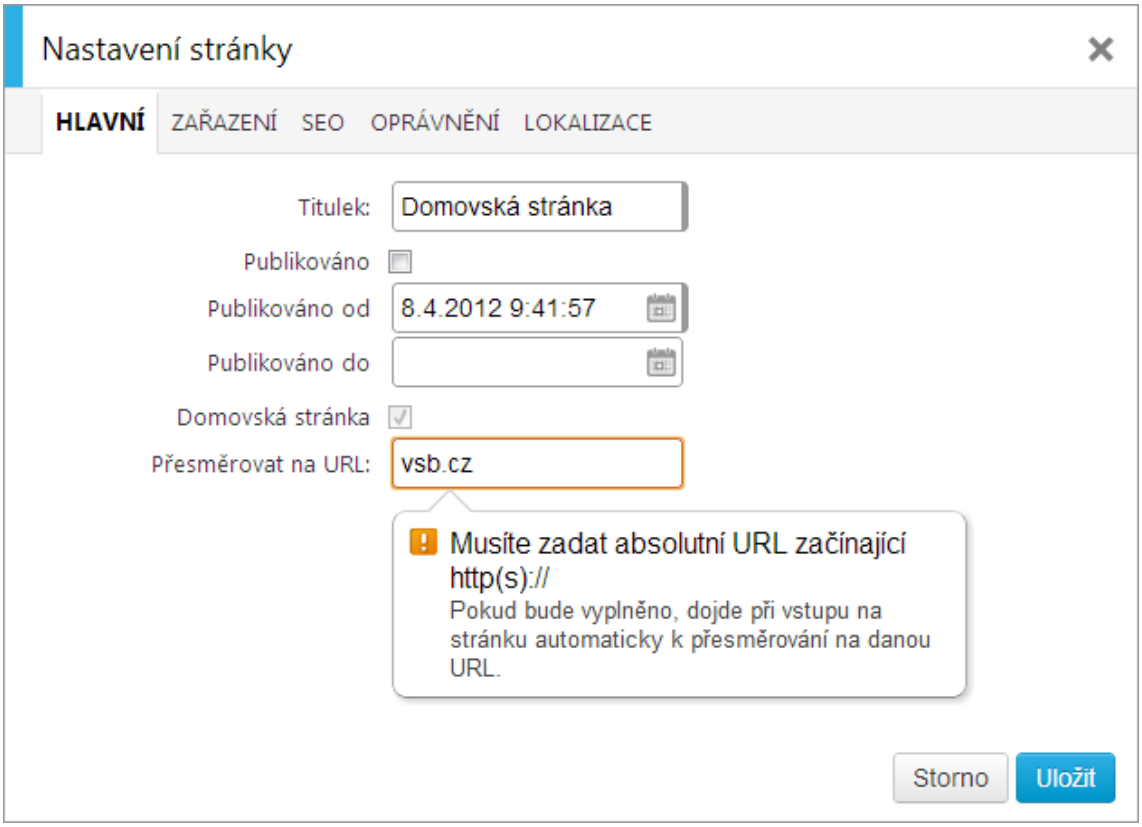
Obrázek 5: Sekvenční diagram životního cyklu formuláře

Životní cyklus začíná požadavkem na formulář, kterým může být například kliknutí na tlačítko pro otevření dialogu s formulářem. Stránka následně předá formulářové komponentě URL, z níž se má formulář načíst, a případně další data, jenž mohou sloužit k vygenerování tohoto formuláře. Komponenta následně provede načtení jeho popisu. Ukázka takovéhoto popisu je na výpise 21. Význam jednotlivých položek je následující:

- **Method** – HTTP metoda, která se má použít pro odeslání formuláře.
- **Enctype** – způsob kódování dat odeslaného formuláře
- **Action** – URL na kterou se má formulář odeslat.
- **FormData** – jednotlivé prvky formuláře v pořadí, ve kterém se mají vykreslit. Důležité je, že není nikde uvedeno, jak se mají jednotlivé prvky vykreslit, nebo validovat. Každá položka má své atributy, jenž závisí na typu položky.
  - **Placeholder** – Zástupný text, který se má zobrazit pokud není vyplněna žádná hodnota
  - **Disabled** – Má být daný formulářový prvek deaktivován?
  - **Description** – Popis formulářového prvku.
  - **Rules** – Seznam definic validačních pravidel.
  - **Attrs** – Seznam HTML atributů, jenž se mají formulářovému prvku přidat.
  - **Type** – Typ formulářového prvku, jenž bude použit pro výběr metody pro jeho renderování.
  - **Name** – Název formulářového prvku. Pod tímto názvem budou data zaslána na serveru.
  - **Label** – Popisek formulářového prvku.
  - **Value** – Hodnota.
- **Identity** – Speciální atribut, jenž má vždy hodnotu „JsonForm“. Slouží ke kontrole zda jsou přijatá data skutečně popisem formuláře, nebo jde například o chybu serveru.
- **Errors** – Může obsahovat seznam chyb validace na straně serveru

```
{
  "FormData":[
    {
      "Key":"title",
      "Value":{
        "Placeholder":null,
        "Disabled":false,
        "Description":"Titulek stránky",
        "Rules":[
          {
            "Type":"FILLED",
            "Message":"Musíte zadat titulek!"
          }, {
            "MaxLength":150,
            "Type":"MAXLENGTH",
            "Message":"Maximální délka titulku je 150 znaků."
          }
        ],
        "Attrs":{},
        "Type":"Text",
        "Name":"title",
        "Label":"Titulek:",
        "Value":"Domovská stránka"
      }
    }, {
      "Key":"save",
      "Value":{
        "Description":null,
        "Disabled":false,
        "Attrs":{},
        "Type":"Submit",
        "Name":"save",
        "Label":null,
        "Value":"Uložit"
      }
    }
  ],
  "Method":"POST",
  "Enctype":null,
  "Action":"/Admin/cs/PageManager/Edit",
  "Identity":"JsonForm",
  "Errors":null
}
```


Výpis 21: Příklad popisu formuláře pomocí JSON



Obrázek 6: Ukázka formulářového dialogu rozděleného do záložek

Po načtení popisu formuláře se provede jeho renderování. Přičemž samotná formulářová komponenta provede renderování pouze úvodního tagu „<form>“ a jeho ukončení. Ostatní formulářové prvky jsou renderovány pomocí k tomu určených metod. Pro standardní HTML prvky jsou tyto metody již součástí komponenty. Pokud je potřeba vytvořit nový (nestandardní) formulářový prvek, stačí u komponenty zaregistrovat příslušnou metodu pro tento typ. Stejně tak je možné výchozí metody přepisovat vlastními. Toto nastavení se provádí na úrovni instance formuláře. Není tedy problém stejný formulář vyrenderovat více způsoby.

Pokud je formulář složitější, bývá zvykem rozdělit jej do tématických skupin tak, aby byl pro uživatele dostatečně přehledný. Díky tomu, že každý formulářový prvek, kterým je v rámci této komponenty i tato skupina prvků, se renderuje pomocí zvolené metody, je možné vykreslit tyto skupiny například jako skupinu záložek, nebo jako akordeon. Takto vyrenderovaný formulář je možné vidět na obrázku 6.

Další fází životního cyklu formuláře je klientská validace uživatelem zadaných dat. Ta se provádí pouze prostřednictvím nástrojů, jenž poskytuje HTML 5. Tyto nástroje již byly zmíněny v sekci „Tagy a sémantika“. Jde například o atributy „required“, „pattern“ a další. Validace těchto prvků probíhá automaticky při odeslání formuláře a provádí j

prohlížeč. Výhod tohoto přístupu, oproti tvorbě vlastní validace pomocí JavaScript-u je několik. Například, že validace funguje i když je JavaScript vypnutý, a nebo, že je validace ve všech prohlížečích vždy korektní. S tímto souvisí i zobrazení chybové hlášky, o kterou se stará také sám prohlížeč.

Pokud daný prohlížeč některé z HTML 5 validačních pravidle nepodporuje, dojde k přeskočení klientské validace a proběhne odeslání dat na server stejně, jako kdyby byla validace úspěšná. Server následně provede opětovnou validaci dat, která je již finální. V rámci této validace je možné kontrolovat i velmi specifická pravidla. Například existence uživatelského jména v databázi a podobně. Pokud není některé z validačních pravidel splněno, je navrácen seznam chyb, jenž je uživateli zobrazen. Celá komunikace probíhá pomocí AJAX-u, takže vše probíhá velmi rychle a tudíž uživatelsky velmi přívětivě.

Standardní součástí všech formulářů vytvořených pomocí této komponenty je ochrana proti útoku CSRF. Podstata tohoto útoku tkví v tom, že korektní uživatel systému navštíví webovou stránku útočníka, a to v době, kdy je tento uživatel přihlášen do systému. Útočník následně provede například pomocí JavaScriptu odeslání formuláře na adresu v rámci systému, a to bez vědomí uživatele. Dík tomu, že je uživatel přihlášen, se tento požadavek jeví jako korektní a je systémem přijat. Tímto je možné například změnit jeho heslo, provést smazání dat a podobně. Pokud je útok proveden korektně, je velmi nebezpečný. Ochrana proti tomuto útoku spočívá v přidání náhodně vygenerovaného řetězce, jenž je spjat s daným uživatelem, do formuláře a jeho uložení na straně serveru. Po odeslání se kontroluje, zda řetězec odpovídá tomu na serveru. Pokud ne, jedná se o pokus o útok. Ochrana je založena na tom, že útočník může pouze odeslat požadavek, ale není schopen načíst nazpět žádná data, tedy ani tento tajný klíč.

Pokud jsou vstupní data validní, jsou na straně serveru data přijata k dalšímu zpracování a na straně klienta je zavolána vývojářem definovaná funkce, pomocí níž je možné například zobrazit správu o úspěšném uložení. Toto je již plně v režii vývojáře.

Na výpise 22 je uveden základní JavaScript-ový kód pro použití formulářové komponenty. A na výpise 23 je ukázka definice formuláře v rámci C#.

```
$('body').JsonForm({
  'url': '/Edit/',
   'data': {
     'id': 1,
   },
  'cancel': function () {
     // Vyplňování formuláře stornováno
   },
  'success': function (data) {
     // Formulář byl úspěšně odeslán a je validní
   }
});
```

Výpis 22: Příklad načtení formuláře v JavaScriptu

```
public ActionResult Edit(int id)
{
  Form form = new Form(Request, CultureInfo.InvariantCulture);
  form.SetAction(Url.Action("Edit"));
  using (new Group(form, "tabsGroup", null, new
     TabsRenderer()))
  {
    using (new Group(form, "mainGroup", "Hlavní"))
    {
      form.AddHidden("id", id.ToString());
      form.AddTextInput("title", "Titulek:")
            .AddRule(new FilledRule("Musíte zadat titulek!"))
            .AddRule(new MaxLengthRule(150, "Maximální délka
               titulku je 150 znaků."))
            .SetDescription("Titulek stránky");
    }
    using (new Group(form, "stateGroup", "Stav"))
    {
      form.AddCheckbox("published", "Publikováno", true)
        .SetDescription("Pouze publikovaná stránka je
           viditelná běžným uživatelům.");
    }
  }

  using (new Group(form, "buttons", new ButtonsRenderer()))
  {
    form.AddSubmit("save", "Uložit");
    form.AddCancel("close", "Storno");
  }
  form.OnSubmit += this.Save;
  return form;
}

private void Save(Form form)
{
  int pageId = (form["id"] as Hidden).ValueAsInt();
  string title = (form["title"] as TextInput).ValueAsString();
  bool isPublished = (form["published"] as
     Checkbox).ValuesAsBool();
  // TODO: Uložit data
}
```

Výpis 23: Příklad definice formuláře

Metoda „Edit“ na výpise 23 je součástí controlleru a je volána jednak pro načtení popisu formuláře, tak pro jeho odeslání. V rámci této metody se vytváří instance třídy „Form“, jenž je základem celého formuláře. Jako parametr je potřeba předat instanci třídy „HttpRequestBase“, z níž si formulář získává data. A jako druhý parametr instanci třídy „CultureInfo“, jenž určuje pravidla pro formát čísel, data a dalších kulturní zvyklosti.

Díky tomu, že třída „Form“ dědí z „ActionResult“, stává se z ní zároveň „View“. Není tedy potřeba vytvářet zvlášť šablonu jen pro renderování formuláře, jako by tomu bylo u standardních ASP.NET MVC formulářů. Výstupem, jenž jde ke klientovi je popis formuláře podobný tomu na výpise 21.

Pomocí metody „SetAction“ se nastaví URL, na kterou se má formulář odeslat.

Jak již bylo uvedeno, formulářová komponenta umožňuje definovat skupiny prvků. K tomuto bylo využito bloku „using“. Ten standardně slouží k uvolnění prostředků, jenž daná třída využívá, ale zde byl využit za jiným účelem, a sice zjednodušení zápisu a zvýšení přehlednosti kódu. Vytvořením instance třídy „Group“ nedělá nic jiného, než že přidá do formuláře nový prvek typu „GROUP_START“. A díky tomu, že daná třída implementuje rozhraní „IDisposible“, je na její instanci při ukončení bloku automaticky zavolána metoda „Dispose“, pomocí níž se do formuláře přidá další speciální prvek typu „GROUP_END“.

Jako jeden z parametrů při vytváření skupiny prvků lze zadat renderer, jímž je každá instance třídy jenž implementuje rozhraní „IRenderer“. Pomocí tohoto rendereru se do popisu formuláře doplní informace o tom, pomocí jaké metody se má daná skupina v rámci JavaScript-u renderovat. V této ukázce je skupin využito k renderování formuláře jako skupiny záložek, podobné jako na obrázku 6.

V rámci jednotlivých skupin jsou již definovány jednotlivé formulářové prvky a jeich validační pravidla. Na obrázku 7 je zjednodušený třídní diagram všech formulářových prvků, jenž jsou standardní součástí komponenty. Pro vytvoření dalších stačí implementovat rozhraní „IControl“.

Poslední věcí, kterou je při definici potřeba udělat, je registrace metody k události „OnSubmit“. Ta je volána v okamžiku, kdy je formulář odeslán a je zároveň validní. Je to tedy ten okamžik, kdy je možné s daty začíst bezpečně pracovat.

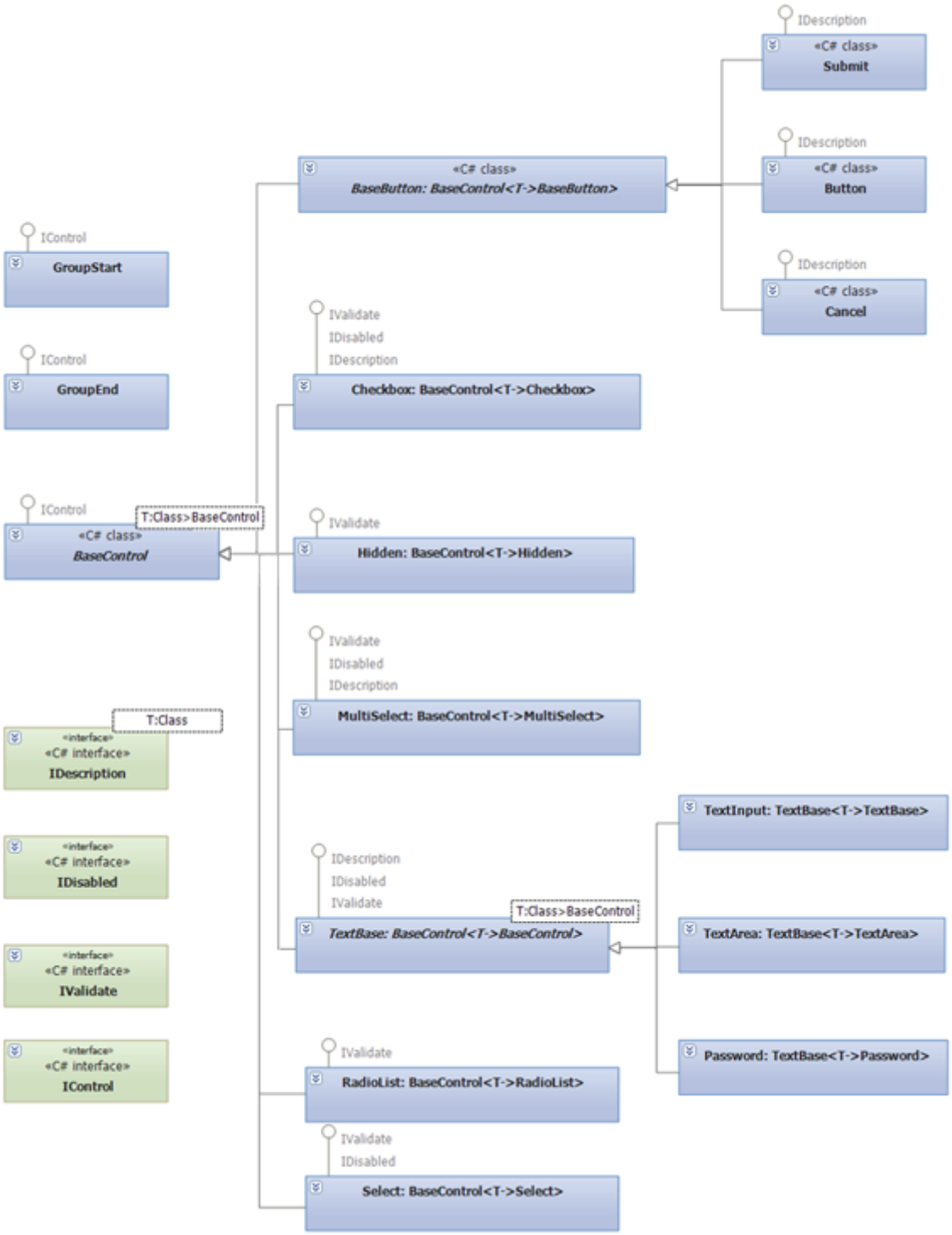
3.4 DataGrid

Předchozí komponenta sloužila čistě pro vstup dat do aplikace. Tato data je ovšem potřeba uživateli nějakým způsobem prezentovat, neboť aplikace, která by přijímala jen vstupy a nedávala žádné výstupy, by nedávala smysl.

Nejpřehlednějším způsobem, jak prezentovat velké množství dat uživateli, je ve formě tabulky. A aby bylo možné se v rámci těchto dat efektivně orientovat, je potřeba uživateli umožnit jejich filtrování a řazení dle různých atributů.

Za tímto účelem vznikla komponenta s názvem „DataGrid“, jenž má usnadnit vytváření právě takovýchto tabulek a to nad různými zdroji dat.

Zjednodušená verze třídního diagramu navržené komponenty je uveden na obrázku 9. Komponenta se skládá z několika částí a to jádra tvořeného třídou „DataGrid“, sloupců, filtrů, datových zdrojů, exportérů, rendererů a tlačítek.



Obrázek 7: Třídní diagram formulářových prvků

3.4.1 Datové zdroje

Základem celé komponenty jsou datové zdroje, jenž slouží k dotazování se na data za pomoci uživatelem definovaných filtrů. Datový zdroj je každá třída, jenž implementuje rozhraní „IDataSource“, jenž definuje dvě metody. První je „GetItemsCount“, pro získání počtů všech položek, jenž odpovídají daným filtrům. A druhou je „GetItems“ pro získání části těchto dat na základě omezení počtu položek na stránku a aktuální stránce.

Součástí komponenty je implementace jednoho z datových zdrojů a sice Microsoft SQL Server-u. Pro jeho zprovoznění je potřeba vytvořit instanci třídy „SqlDataSource“, jejíž konstruktor vyžaduje přihlašovací řetězec k databázi a název primárního klíče tabulky, na níž se bude dotazovat. Ten je potřeba kvůli efektivnímu stránkování, jenž SQL Server na rozdíl třeba od Oracle umožňuje. Následně se instanci této třídy předá základní SQL dotaz na všechna data. Datový zdroj si tento SQL dotaz, při následném dotazování, již sám zpracuje a aplikuje na něj veškeré uživatelem definované filtry a stránkování. Na výpise 24 je uvedena ukázka dotazu vygenerovaného na základě základního dotazu „SELECT * FROM [ModuleAssembly]“, na nějž je aplikován filtr a stránkování. Dotaz může na první pohled vypadat neefektivně, neboť základní dotaz vybírá všechny data z dané tabulky. Ovšem databázový optimalizátor SQL Server-u je na tyto dotazy připraven a je schopen si je upravit a provést skutečně efektivně.

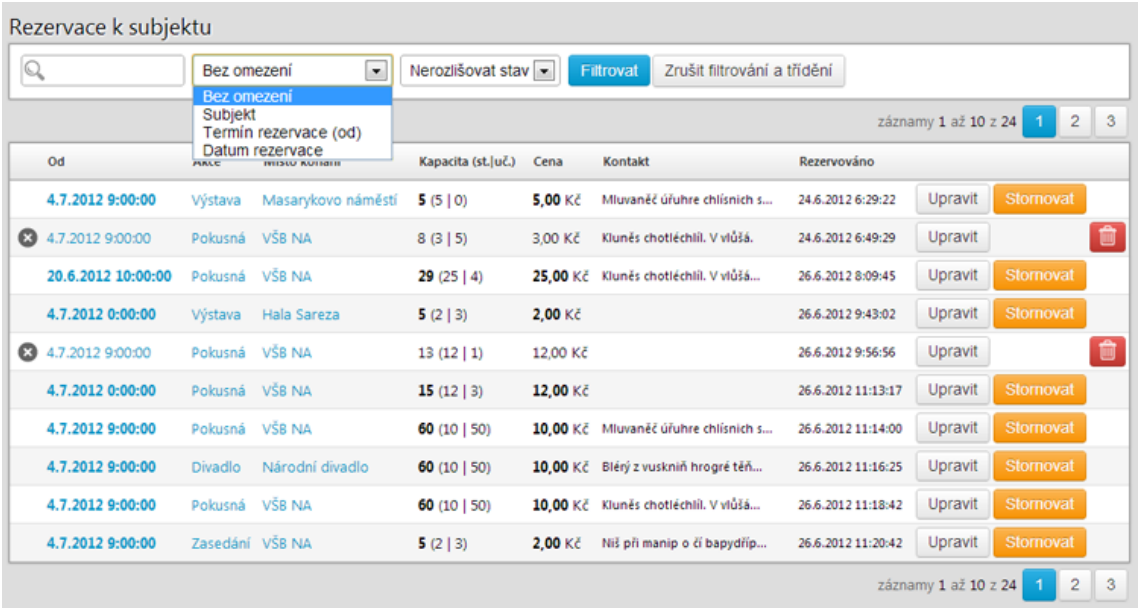
```
SELECT [table2TmpName9245201245].* FROM (
  SELECT ROW_NUMBER() OVER (
    ORDER BY [table1TmpName9245201245].[Id]
  ) AS [RowNumber],
  [table1TmpName9245201245].* FROM (
    SELECT * FROM [ModuleAssembly]
  ) AS [table1TmpName9245201245] WHERE (
    [table1TmpName9245201245].[Directory] LIKE @Var0
  )
) AS [table2TmpName9245201245]
  WHERE [RowNumber] BETWEEN 1 AND 25;
```

Výpis 24: Příklad vygenerovaného SQL dotazu datovým zdrojem

Data z datového zdroje jsou vždy vráceny jako seznam objektů typu „Record“. Každý „Record“ obsahuje data jedné entity jako slovník, kde klíčem je název sloupce. Důležité je, že tento název sloupce nemusí odpovídat názvu v rámci datového uložiště (například databáze). Název může být určen až při mapování, což navíc umožňuje přidání dalších sloupců, jenž mohou být například vypočteny, nebo pocházet z jiného zdroje.

3.4.2 Sloupce a tlačítka

Sloupec je v rámci DataGridu každá třída, jenž implementuje rozhraní „IColumn“. Samotný sloupec pak slouží k popisu sloupců viditelných uživateli (nejedná se o sloupce v rámci datového zdroje) a jejich buněk. Rozhraní „IColumn“ obsahuje tři property. První



Obrázek 8: Ukázka výstupu komponenty „DataGrid“

udává název sloupce, jenž byl namapován z datového zdroje a jeho hodnoty budou vloženy do buněk daného sloupce. Další property je titulek sloupce viditelný uživateli a poslední je název CSS třídy, pro snazší aplikování stylů. Rozhraní dále vyžaduje implementaci metody pro formátování hodnot jednotlivých buněk daného sloupce.

Metoda pro formátování dat zde hraje velmi důležitou úlohu. Načtená data z datového zdroje mohou být v různých formátech, ať už jde o datum, číslo, řetězec a další, jenž je potřeba zobrazit uživateli v určitém formátu. A také může jít o speciální data, jenž je potřeba vizualizovat jinak než jejich hodnotou. Příkladem takových dat může být název obrázku, který chceme zobrazit, nebo nějaká výčtová hodnota, kterou je potřeba nahradit za její uživatelsky přívětivou alternativu.

Na obrázku 8 lze vidět výstup s různě naformátovanými sloupci.

Z třídního diagramu na obrázku 9 lze vyčíst, že součástí komponenty jsou tři základní typy sloupců. „Column“, jenž převede hodnotu na textový řetězec a provede escapování nebezpečných znaků. Sloupce „DateTimeColumn“, jak název napovídá, slouží k formátování data do textové podoby. A „ActionColumn“ je speciálním sloupcem, jenž umožňuje přidávat tlačítka, které lze použít například na editaci záznamů.

Každé tlačítko, které je přidáno do sloupce „ActionColumn“, musí dědit z abstraktní třídy „ActionButtonBase“. Jedinou úlohou těchto tlačítek je provést renderování příslušného HTML kódu. Ten by bylo samozřejmě možné vytvářet i ručně v rámci metody pro formátování sloupce, ale to by znamenalo omezení v rámci znovupoužitelnosti a rozšiřitelnosti.

3.4.3 Filtry

Filtr je v rámci komponenty DataGrid každá třída, jenž implementuje rozhraní „IFilter“. Jejich úkolem je na základě zadaných hodnot uživatele vyfiltrovat požadované záznamy. Každý z filtrů je sám zodpovědný za načítání dat pro filtrování z aktuální URL a stejně tak za jejich serializaci nazpět do URL generovaných DataGridem. Díky tomuto je zajištěno, že při stránkování DataGridu, změně řazení položek a podobně, nedojde k zrušení nastavených filtrů.

Součástí komponenty jsou, jak lze vyčíst z třídního diagramu na obrázku 9, dva filtry. Prvním je „TextFilter“, jenž se skládá z textového pole pro zadání hledaného výrazu a roletky pro výběr sloupce, v rámci nějž se má vyhledávat. Při definici těchto sloupců je potřeba určit datový typ daného sloupce v databázi, což umožňuje korektní parsování vstupních dat. Díky čemuž je možné vyhledávat například podle části data, jako je rok, měsíc v roce a podobně.

Druhým z filtrů je „SelectFilter“, jenž slouží k filtrování podle nějakého výčtového typu. Uživatel nemá možnost zadat ručně hodnotu, ale může zvolit z daných hodnot, jako například „Stornované rezervace“, „Aktivní rezervace“ a podobně.

Ukázka filtrů je na obrázku 8.

3.4.4 Renderery a Exportéry

Renderery slouží k vykreslení komponenty jako celku. Pro vytvoření rendereru je potřeba implementovat rozhraní „IRenderer“. Toto oddělení vykreslování od logiky komponenty umožňuje nejen jednoduchou změnu vzhledu komponenty, ale také renderování do jiných formátů, než je HTML. Přímo součástí komponenty je renderer, jenž vytvoří soubor ve formátu Microsoft Excel (*.xlsx). Stejně tak by bylo možné provést renderování do XML, nebo dokonce generování obrázku. S tomto směru není možnost rozšíření nijak omezená.

Poslední součástí komponenty jsou exportéry. Ty jsou velmi podobné rendererům. Hlavní rozdíl je v tom, že rendererům jsou předána data, jenž podléhají všem aktuálně nastaveným filtrům a stránkování. Kdežto exportéry mají možnost přístupu ke všem datům a mohou si je samy v případě potřeby filtrovat. Pro vytvoření exportéru je potřeba implementovat rozhraní „IExporter“. Součástí komponenty je exportér do formátu Microsoft Excel (*.xlsx).

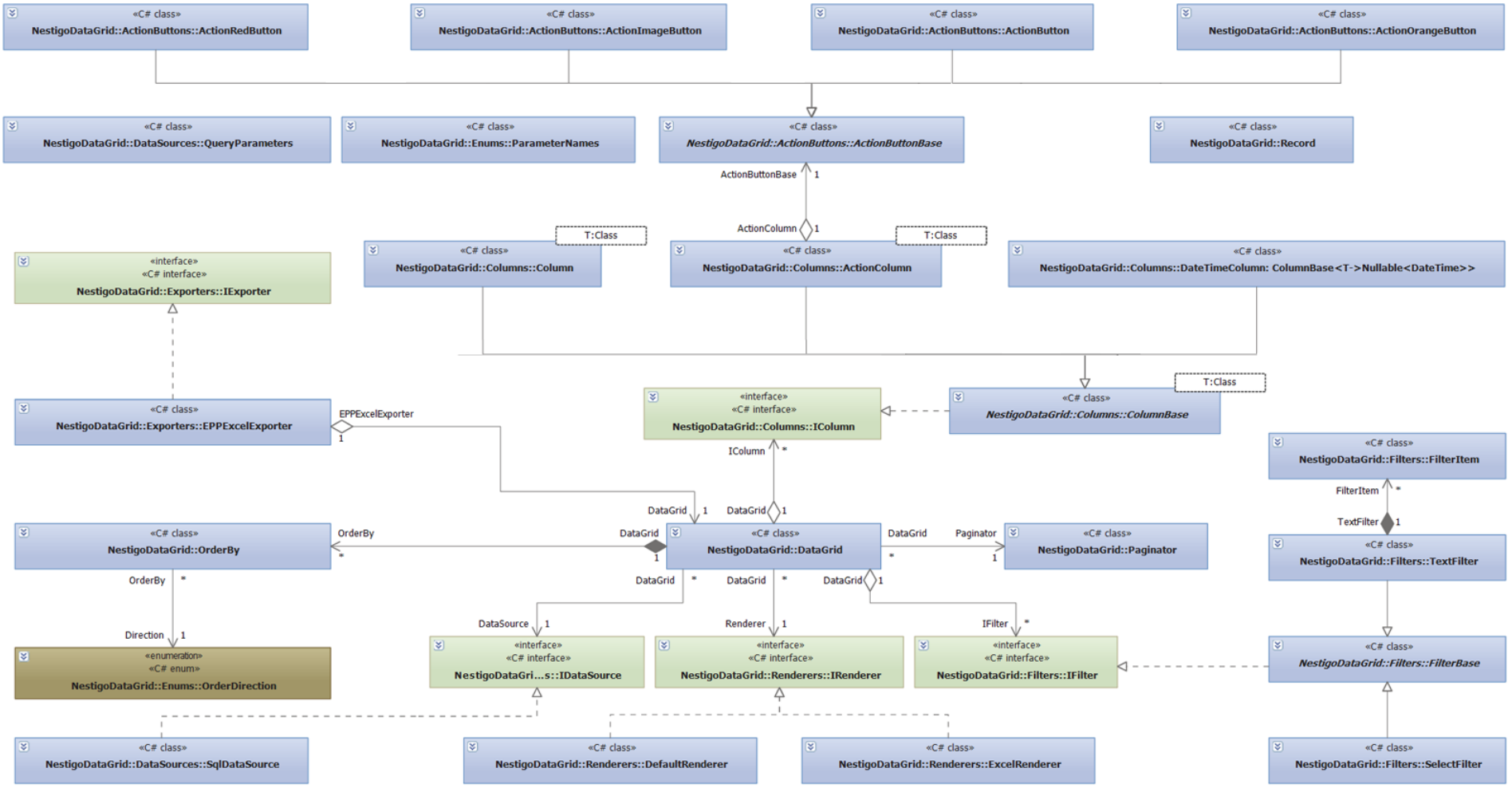
3.4.5 Příklad použití

```
SqlDataSource dataSource = new SqlDataSource(ConnectionString,
    "Id") {
  Command = "SELECT * FROM [Reservations]",
  Select = delegate(SqlDataReader reader){
    return new Record().Read(reader)
  }
};
```

```
DataGrid grid = new DataGrid(ControllerContext, "grid");
grid.DataSource = dataSource;
grid.AddColumn(new Column<bool>("IsCanceled", "",
   delegate(bool value, Record recod)
{
  return new HtmlString(value ? "<img src=\"canceled.png\"
     alt=\"Stornováno\" />" : "");
}, "iconshover"));
grid.AddColumn(new DateTimeColumn("Start", "Od"));
grid.AddColumn(new Column<decimal>("PriceSum", "Cena",
   delegate(decimal value, Record record){
  string s;
  if(bool)record["IsCanceled"] == false){
    s = "<strong>" + value + "</strong>";
  }else{
    s = value.ToString()
  }
  return new HtmlString(s + " Kč");
}));
grid.AddFilter(new TextFilter("Filter1", new List<FilterItem>()
{
  new FilterItem("Start", "Termín rezervace (od)",
     DbType.DateTime),
   new FilterItem("PriceSum", "Cena", DbType.Decimal)
}));
ActionButton btn = new ActionButton("Upravit", x =>
   Url.Action("Edit", "Event", new { Id = x["Id"]}));
grid.AddColumn(new ActionColumn<int>("Id", "", "action", btn));
grid.Render();
```

Výpis 25: Příklad použití komponenty DataGrid

Kód ukázky by měl být již jasný z předchozího popisu komponenty. Jediné co ještě nebylo zmíněno je property „Select“ u datového zdroje SQL Server-u. Tato property slouží k mapování dat z databáze (prostřednictvím standardního SqlDataReader-u) na objekt „Record“, o němž již byla řeč. Toto mapování je možné provést buď to ručně, nebo jako v ukázce pomocí metody „Read“ třídy „Record“, jenž jej provede automaticky. Hlavní rozdíl mezi ručním a automatickým mapováním je v tom, že to automatické namapuje všechny atributy, které byly z databáze načteny, a to pod názvy, jako v databázi. Kdežto pomocí ručního mapování je možné názvy libovolně měnit a mapovat jen některé atributy.



Obrázek 9: Třídní diagram pro NestigoDataGrid

3.5 Vizualizace grafů s využitím hardwarově akcelerované grafiky

Standard HTML 5, mimo jiné, přichází s rozšířením možností práce s grafikou na webu. Jedná se o vektorovou grafiku ve formátu SVG, jenž je založen na formátu XML. A dále o rastrovou grafiku, jenž lze generovat v rámci nového HTML elementu „canvas“.

Pro účely zamýšlené vizualizace grafů se tento text omezí pouze na práci s rastrovou grafikou. Hlavním důvodem, proč nevyužít vektorovou grafiku je náročnost jejího zpracovávání a s tím spojený nízký výkon aplikace, obzvláště s rostoucím počtem vykreslovaných prvků. Důvodem k tomuto nízkému výkonu, i přes hardwarovou akceleraci vykreslování, je samotný způsob popisu SVG formátu. Jde o XML dokument, jenž je v rámci prohlížeče reprezentován pomocí DOM. Ten je před samotným vykreslením nutno vybudovat, což může být u rozsáhlých dokumentů velmi náročná operace. Pro vizualizaci grafů je navíc potřeba často měnit pozici vrcholů a hran, což vede ke změnám v rámci DOM. Provádění těchto změn je pomalé a ne zcela efektivní. Na druhou stranu v okamžiku kdy je na základě daného DOM vygenerován výstup, a nedojde k jeho změně, není nutné jej znovu generovat.

Canvas je na tom s výkonem, při vykreslování rozsáhlé často se měnící scény, o poznání lépe. Důvodem jsou lepší možnosti hardwarové akcelerace. Kreslení do elementu canvas probíhá pomocí JavaScript-ového API, jenž se liší dle zvoleného kontextu. Ten může být buď to „2D“, nebo „3D“.

3.5.1 Canvas 2D

Canvas 2D je označení kontextu canvasu, jenž umožňuje kreslení v rámci dvourozměrného prostoru. JavaScript-ové API obsahuje metody pro kreslení elementárních prvků scény jako jsou čáry, obdélníky a podobně. Na rozdíl od SVG tedy neexistuje žádný popis scény, na základě kterého by došlo k vykreslení. Kreslení probíhá na základě volání metod API a to ve vrstvách, podobně jako když malíř kreslí obraz.

Důvod, proč je kreslení komplikovaných a často se měnících scén efektivnější, než-li v rámci SVG, je podobnost příkazů s příkazy grafických knihoven, jako jsou OpenGL, nebo DirectX. Ty jsou dnes již standardně implementovány na úrovni hardwaru. Při volání metody v rámci JavaScript-ového API tedy stačí, aby prohlížeč převedl daný příkaz na ekvivalentní v rámci grafické knihovny, a ten spustil přímo na grafické kartě. Některé z příkazů, jako je například kreslení čar, je navíc možné hromadit v paměti a následně poslat na grafickou kartu, která je dokáže zpracovat v rámci jediného příkazu.

Tento převod na příkazy grafické karty je pochopitelně možný i v rámci SVG, ale u něj je navíc potřeba pracovat s DOM.

```
var canvas = document.createElement("canvas");
document.body.appendChild(canvas);
canvas.width = 800;
canvas.height = 600;
var ctx = canvas.getContext("2d");
ctx.fillStyle = "#3cafe2";
ctx.strokeStyle = "#ffffff";
```

```
ctx.lineWidth = 6;
ctx.lineCap = "square";
ctx.fillRect(0, 0, 800, 600);
ctx.beginPath();
ctx.moveTo(50, 50);
ctx.lineTo(750, 50);
ctx.lineTo(50, 550);
ctx.lineTo(50, 50);
ctx.stroke();
```

Výpis 26: Příklad použití Canvas 2D

Na výpise 26 je uveden příklad toho jak pracovat s 2D kontextem canvasu. Nejprve je potřeba vytvořit HTML element canvas a vložit jej do stránky. Následně je potřeba nastavit šířku a výšku kreslící plochy. Tyto rozměry mohou, ale nemusí, odpovídat rozměrům elementu canvas na stránce. Změnou poměru rozměrů elementu k rozměrům kreslící plochy lze dosáhnout změny dpi.

Posledním krokem před kreslením je získání samotného 2D kontextu pomocí metody „getContext“.

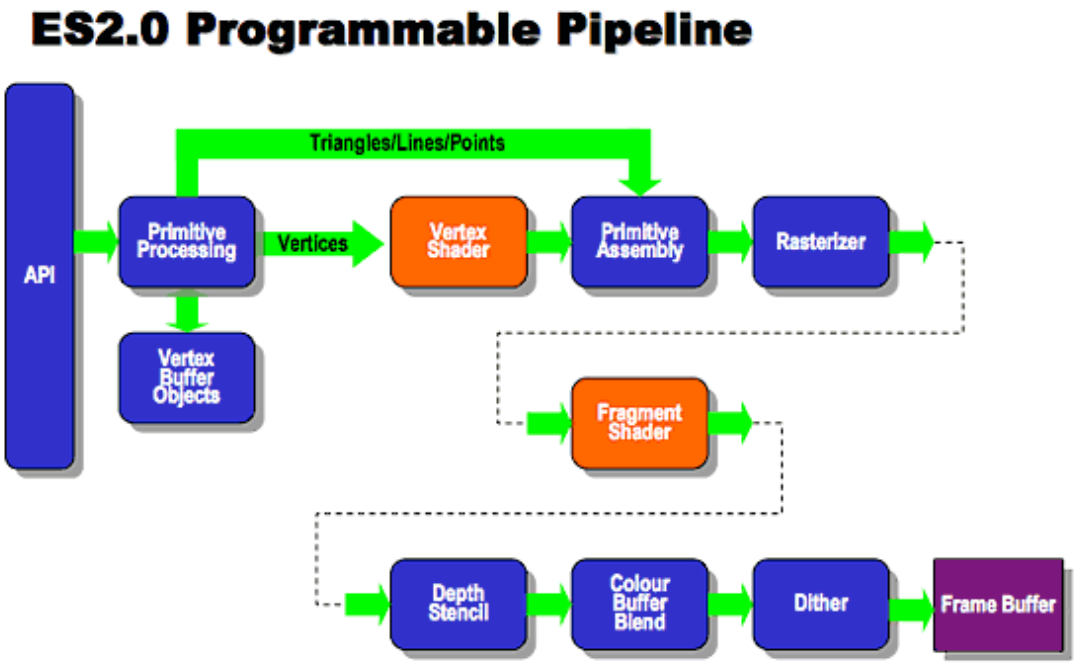
Canvas 2D, podobně jako třeba OpenGL, funguje jako stavový stroj. Neboli v okamžiku, kdy se mu nastaví nějaký stav, setrvává v tomto stavu až do chvíle, než jej opět změníme. Jelikož je změna těchto stavů náročná, je vhodné vykreslovat prvky v takovém pořadí, aby bylo potřeba co nejméně změn stavu. V ukázkovém příkladě je nastavení celkem čtyř stavů a to barva výplně, barva čáry, její tloušťka a typ zakončení.

Metoda „fillRect“ slouží k nakreslení obdélníku, jenž v tomto případě tvoří pozadí. Metoda „beginPath“ slouží k začátku dávky kreslících příkazů, jenž mají být interpretovány jako jeden celek. Toto je důležité obzvláště pokud se bude následně aplikovat barevná výplň. Bez použití „beginPath“ by se aplikovala na všechny doposud nakreslené prvky. K metodě „beginPath“ ovšem neexistuje žádná párová metoda „endPath“. Při volání „beginPath“ dojde interně k ukončení stávajícího zásobníku příkazů a vytvoří se nový. Jakýkoliv kreslící příkaz musí patřit do některého ze zásobníků. Z tohoto by mělo být zřejmé, že i bez ručního zavolání metody „beginPath“ před začátkem kreslení, je tato metoda volána automaticky. A také, že není potřeba zásobník příkazů nijak ukončovat.

Pomocí metod „moveTo“ a „lineTo“ dochází k definici vrcholů čáry, jenž se vykreslí, až v okamžiku zavolání metody „stroke“ na posledním řádku. Z tohoto způsobu kreslení čáry by mělo být jasné, jakým způsobem dochází k hardwarové akceleraci této operace. Příkazy „moveTo“ a „lineTo“ slouží pouze k vybudování pole vrcholů, jenž je až po zavolání „stroke“ zasláno na grafickou kartu a tam vykresleno.

3.5.2 Canvas 3D (WebGL)

Canvas 3D, spíše označovaný jako WebGL, je kontext canvasu, jenž umožňuje kreslení v rámci trojrozměrného prostoru. Pro kreslení je opět využito JavaScript-ové API, které až na pár drobných odlišností kopíruje specifikaci standardu OpenGL ES 2.0. Toto umožňuje nejlepší hardwarovou akceleraci ze všech uvedených metod.



Obrázek 10: Pipeline OpenGL ES 2.0 (Zdroj: „http://www.khronos.org/opengles/2_X/“)

Důvodem, proč byl zvolen právě standard OpenGL ES 2.0 je jeho jednoduchost, snaha přesunout většinu funkcionality na úroveň shaderů a co nejvíce omezit přenos dat mezi aplikací a grafickou kartou. Toto je z pohledu webu velmi důležité. Je potřeba si uvědomit, že programovacím jazykem je zde JavaScript. Tedy dynamický interpretovaný scriptovací jazyk. Jako takový je ve srovnání s jazyky jako C, nebo C++, velmi pomalý a hlavně neumožňuje přímý přístup k hardware. Snaha přenést maximum práce na grafickou kartu je tedy logická.

Ačkoliv WebGL staví na standardu OpenGL, není na většině počítačů s operačním systémem Windows podpory OpenGL využíváno. Místo toho dochází k překladu OpenGL příkazů na příkazy DirectX. Prohlížeče Chrome a Firefox k tomuto využívají projekt ANGLE (Almost Native Graphics Layer Engine). Opera obsahuje vlastní překladač. Na operačních systémech postavených na linuxu je naopak standardně využíváno OpenGL. Pokud zařízení nedisponuje grafickou kartou, nebo grafická karta nepodporuje DirectX, nebo OpenGL, je využito softwarového renderování.

WebGL, jak již bylo zmíněno, se snaží přesunout většinu funkcionality do shaderů. S tímto souvisí zjednodušená pipeline, jenž je na obrázku 10. Transformace a výpočet osvětlení zde probíhá v rámci vertex shaderu, jenž provádí výpočty nad jednotlivými vrcholy a jeho povinným výstupem je transformovaná pozice těchto vrcholů.

Texturování, výpočet barvy, aplikování mlhy a další efekty jsou zde nahrazeny fragment shaderem. Ten provádí výpočty nad jednotlivými pixely, výsledné scény. Jeho povinným výstupem je barva daného pixelu.

Pro definici shaderů je v rámci WebGL možno využít jazyk GLSL ES verze 1.0, jenž vychází z jazyka GLES 1.4, který je součástí OpenGL 3.1. Zřejmě nejdůležitější změnou oproti předchozím verzím GLES je nutnost vlastní definice projekčních, transformačních, normálových a případně dalších matic, jenž nejsou v rámci vertex shaderu standardně dostupné.

Další důležitou vlastností WebGL je neexistence tzv. přímého módu. Jde obecně o příkazy, jenž se vyžadují funkce glBegin() a glEnd(). Tedy například glVertex, glColor, glNormal, glTexCoord, glIndex, glMaterial a další. Důvodem, proč tyto funkce neobsahuje, je opět výkon. Volání funkce pochází z JavaScript-u a je ho potřeba společně se vstupními daty přeposlat na grafickou kartu, což je velmi pomalá operace. Místo toho je preferováno využití ArrayBuffer-u z něhož je následně možné vytvořit Vertex Object Buffer.

Zmíněný „ArrayBuffer“ je jedním z nových objektů v rámci JavaScriptu, jenž přichází společně s HTML 5. ArrayBuffer je generický buffer binárních dat s pevně danou délkou, který slouží jako základ typových polí. S jeho obsahem nelze přímo manipulovat. K této činnosti je potřeba využít právě některé z typových polí, jenž jsou rozšířením tohoto objektu a tvoří rozhraní, jenž umožňuje pracovat s binárními daty, jako s polem daného datového typu. Definována jsou tato typová pole:

- Int8Array - 8-bitový integer se znaménkem
- Uint8Array - 8-bitový integer bez znaménka
- Uint8ClampedArray - 8-bitový ořezaný integer bez znaménka
- Int16Array - 16-bitový integer se znaménkem
- Uint16Array - 16-bitový integer bez znaménka
- Int32Array - 32-bitový integer se znaménkem
- Uint32Array - 32-bitový integer bez znaménka
- Float32Array - 32-bitové desetinné číslo dle IEEE754
- Float64Array - 64-bitové desetinné číslo dle IEEE754

Práce s poli je velmi jednoduchá. Například nové pole int-ů se vytvoří pomocí „*new Int32Array(10);*“, kde číslo v závorkách udává délku pole. Stejně tak je možné vytvořit typové pole na základě již existujícího netypového pole. A to tak, že místo délky pole se v závorkách předá stávající pole, nad kterým dojde automaticky k přetypování. Takovéto typové pole lze následně využít v rámci WebGL.

Práce s WebGL je totožná jako v jiných jazycích (například C++). Všechny metody a jejich parametry jsou totožné, pouze se volají na 3D kontextu daného canvasu.

3.5.3 Layoutovací algoritmy založené na výpočtu sil

Algoritmy založené na výpočtu sil (Force-based algoritmy) je skupina algoritmů, jenž slouží k uspořádání uzlů v grafu takovým způsobem, aby je bylo možné esteticky vykreslovat. Tyto algoritmy se obecně snaží o to, aby se jednotlivé uzly nepřekrývaly a jejich geometrická vzdálenost zhruba odpovídala té grafové.

Základem všech force-based algoritmů je **energetický model** a **minimalizační funkce**. Energetický model popisuje způsob jak vypočítat energii v rámci grafu a minimalizační funkce se snaží množství této energie minimalizovat.

Energetický model je tvořen pomocí několika typů sil, jenž se liší dle zvoleného modelu. Základem většiny modelů jsou dvě síly, a to repulzní (repulsion) a přitažlivá (attraction). Velmi často se v modelech objevuje i gravitační síla (gravitation).

Jedním ze základních force-based algoritmů je **Fruchterman-Reingold**. Energetický model je zde popsán pomocí soustavy pružin, jenž reprezentují jednotlivé hrany grafu. Základní myšlenkou algoritmu je, že se všechny uzly navzájem odpuzují a hrany se naopak snaží jednotlivé dvojice držet pohromadě.

Algoritmus běží v iteracích, během nichž vypočte pro každý z uzlů repulzní a přitažlivé síly, jenž na daný vrchol působí. A následně (až poté co jsou pro všechny uzly vypočteny příslušné síly) všechny vrcholy posune o hodnotu síly jenž na ně působí.

Výpočet repulzní síly mezi vrcholy se provádí dle vzorce (1). Tuto sílu je potřeba vypočíst mezi každou dvojicí uzlů, nepříjemně ovlivňuje celkový výkon algoritmu. „C“ a „K“ ve vzorci jsou konstanty. „K“ udává optimální vzdálenost mezi vrcholy a ovlivňuje pouze vzhled (velikost) výsledného grafu. Konstanta „C“ umožňuje relativně zvýšit, či snížit, význam repulzivní síly oproti přitažlivé.

Z uvedeného vzorce je patrné, že velikost repulzní síly klesá s rostoucí vzdáleností a naopak.

$$f_r(i, j, x, K, C) = \frac{-CK^2}{\|x_i - x_j\|}; i \neq j; i, j \in V \tag{1}$$

Výpočet přitažlivé síly mezi vrcholy se provádí dle vzorce (2). Síla se počítá pouze mezi dvojicemi vrcholů, jenž jsou spojeny hranou. Velikost přitažlivé síly klesá s klesající vzdáleností a naopak.

$$f_a(i, j, x, K, C) = \frac{\|x_i - x_j\|^2}{K}; i \leftrightarrow j; i, j \in V \tag{2}$$

Výsledná síla působící na i-tý vrchol se vypočte dle vzorce (3).

$$f(i, x, K, C) = \sum_{i \neq j} \frac{-CK^2}{\|x_i - x_j\|^2}(x_i - x_j) + \sum_{i \leftrightarrow j} \frac{\|x_i - x_j\|}{K}(x_i - x_j) \tag{3}$$

Množství energie v grafu se následně vypočte jako součet energií působících na jednotlivé uzly.

Velikost posunu vrcholu se vypočte dle vzorce (4). Kde „step“ je velikost posunu, jenž může být konstantní, nebo se snižovat s klesající energií v grafu, čímž lze dosáhnout lepších výsledků.

$$x_i = x_i + step * \frac{f(i, x, K, C)}{\|f(i, x, K, C)\|} \tag{4}$$

Počet iterací, jenž bude potřeba pro minimalizaci energie nelze předem pevně stanovit. Zároveň nelze přesně určit okamžik, kdy by měl algoritmus ukončit svou činnost. Algoritmus by měl skončit v okamžiku, kdy je energie v grafu minimální. Tuto hodnotu ovšem neznáme. Pro ukončení algoritmu lze využít toho, že množství energie začne po čase konvergovat k nějaké hodnotě a už se příliš nemění. Toto ovšem platí převážně u malých řídkých grafů. Kromě tohoto omezení je tedy vhodné zavést omezení na maximální počet iterací, či jiná omezení, jež se budou snažit nějakým způsobem detekovat ideální okamžik k ukončení činnosti algoritmu.

3.5.4 Barnes-Hut algoritmus

Při popisu „Fruchterman-Reingold“ algoritmu bylo uvedeno, že se repulzní síly počítají pro každou dvojici vrcholů. Stejně tak tomu je u většiny těchto algoritmů. Časová složitost těchto algoritmů je díky tomu při nejmenším $O(|V|^2)$.

Vzhledem k tomu, že cílem layoutovacích algoritmů není nalézt rozložení při němž je energie v grafu minimální, ale pouze nalézt esteticky přívětivé rozložení vrcholů (minimalizace energie je pouze nástroj jak tohoto rozložení dosáhnout), je možné přistoupit k aproximaci hodnot energie a tím snížit časovou náročnost.

Algoritmus, který tuto aproximaci umožňuje, se nazývá **Barnes-Hut**. Základní myšlenkou tohoto algoritmu je rozdělit prostor na menší části, které za určitých podmínek mohou při výpočtu sil nahradit vrcholy ležící uvnitř tohoto prostoru.

Dělení prostoru probíhá pomocí „quadtree“ v rámci 2D prostoru a „octree“ v rámci 3D prostoru. Tento text se omezí pouze na quadtree.

3.5.4.1 Quadtree Jde o stromovou strukturu, kde každý uzel buď'to dělí plochu na čtyři stejně velké kvadranty, nebo obsahuje maximálně čtyři hodnoty, jenž leží v kvadrantu, jenž uzel reprezentuje. Při konstrukci stromu se nejprve vypočte minimální bounding box. Tedy nejmenší oblast v níž leží všechny hodnoty. Rozměry bounding boxu se následně upraví tak, aby tvořil čtverec, nad kterým bude algoritmus pracovat. Začnou se postupně vkládat hodnoty.

Při vkládání se začne kořenem. Pokud není rozdělen na kvadranty a obsahuje méně než čtyři hodnoty, dojde k vložení.

Pokud není rozdělen na kvadranty, ale obsahuje již čtyři hodnoty, dojde k rozdělení na kvadranty a přesunu všech uzlů, které obsahoval, do příslušných kvadrantů (ve kterých geometricky leží) a aktuálně vkládaná hodna se také přesune do příslušného kvadrantu a popsaný proces postupuje rekurzivně.

Pokud je uzel již rozdělen na čtyři kvadranty, přejde se na příslušný kvadrant a pokračuje se rekurzivně.

Pokud je dělená plocha velká a hodnoty jsou blízko u sebe, může dojít k přetečení zásobníku z důvodu velké hloubky rekurze. Je tedy potřeba zavést omezení na maximální hloubku rekurze a definovat, co se má v takovém okamžiku stát. Pro účely layoutovacího algoritmu se jeví jako nejlepší hodnotu posunout tak, aby ji náhodně, tak aby ji bylo možné vložit.

Konstrukce a procházení tímto stromem má časovou složitost pouze O(| V | log | V |)

3.5.4.2 Aproximace Jednotlivé vrcholy grafu jsou nyní rozděleny do kvadrantů v rámci quadtree. Tyto kvadranty nyní budou sloužit jako náhrada za vrcholy, jenž v nich leží. Nebude tedy potřeba počítat repulzní síly mezi všemi dvojicemi vrcholů, ale bude možné vypočítat repulzní sílu mezi vrcholy a kvadranty, jenž můžou obsahovat velké množství vrcholů. Tím se dramaticky sníží počet nutných výpočtů.

Za tímto účelem je nejprve vytvořit tzv. **super-uzel**. Super-uzel, je uzel jenž leží v těžišti každého z kvadrantů quadtreea, a má váhu rovnu součtu vah všech potomků tohoto kvadrantu. Pro výpočet parametrů těžiště slouží vzorce (5), (6) a (7). Parametr „j“ udává počet potomků aktuálního kvadrantu (ať už vrcholů, nebo super-uzlů). „m“ je váha a „x“ a „y“ pozice. Index „s“ označuje parametry super-uzlu.

$$m_s = \sum_{i=0}^{j} m_i \tag{5}$$

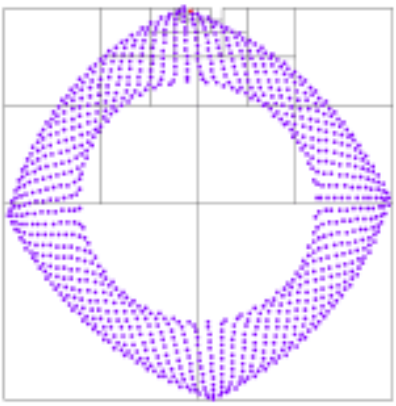
$$x_s = \frac{\sum_{i=0}^{j} (x_i * m_i)}{m_s} \tag{6}$$

$$y_s = \frac{\sum_{i=0}^{j} (y_i * m_i)}{m_s} \tag{7}$$

Nyní lze přistoupit k samotnému výpočtu sil, respektive jejich aproximaci pomocí super-uzlů. Tu je možné použít pokud je super-uzel dostatečně daleko od vrcholu na kterém se síla počítá. To, zda je super-uzel dostatečně daleko, se určí pomocí vzorce (8), který je podmínkou pro aproximaci. θ je konstanta. Čím více se bude θ blížit k nule, tím méně se bude využívat aproximace a výpočet bude náročnější (při hodnotách kolem nuly se aproximace nebude provádět vůbec a výsledná časová složitost bude dokonce vyšší, než kdyby se Barnes-Hut algoritmus nepoužil). Při velkém θ se naopak bude zhoršovat kvalita výstupu. Dobrým kompromisem mezi výkonem a kvalitou výstupu je hodnota θ okolo 1,2.

$$\frac{d_S}{\|x_i - x_S\|} \leqq \theta \tag{8}$$

Proměnná „d_S“ definuje šířku kvadrantu k němuž přísluší aktuální super-uzel. A „$\|x_i - x_S\|$“ je vzdálenost vrcholu od super-uzlu. Pokud je super-uzel dostatečně daleko od vrcholu, provede se výpočet síly pouze mezi ním a vrcholem. Pokud není dostatečně



Obrázek 11: Aproximace pomocí Barnes-Hut algoritmu (Zdroj: „http://www2.research.att.com/ÿifanhu/PUB/graph_draw_small.pdf“)

daleko, přejde se o úroveň níže a vše se opakuje vzhledem ke všem potomkům. Výběr super-uzlů se provádí vždy od kořene stromu směrem k listům. Pokud není super-uzel dostatečně daleko, provede se výpočet vůči jednotlivým vrcholům, které tvoří listy stromu.

Na obrázku 11 je znázorněn klesající počet výpočtů se vzdáleností od vrcholu na nějž se síla počítá.

3.5.5 Víceúrovňový algoritmus

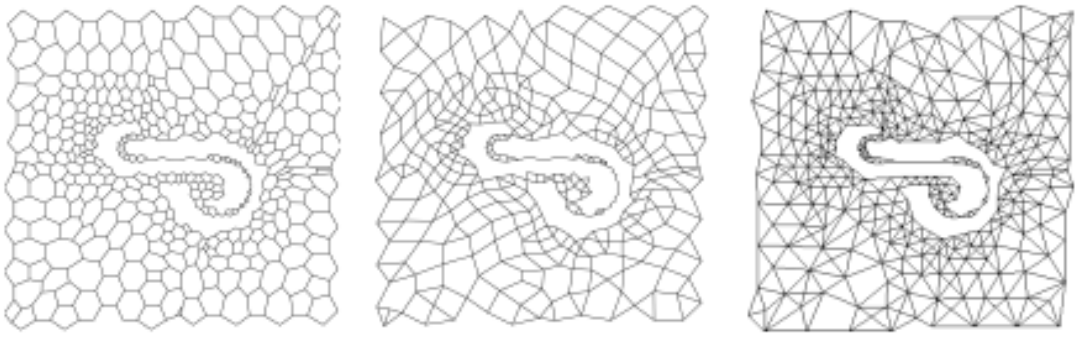
Pomocí Barnes-Hut algoritmu popsaného v předchozí kapitole bylo možné snížit časovou složitost z $O(|V|^2)$ na $O(|V|log|V|)$. Ovšem u opravdu velkých grafů není toto snížení dostačující. Navíc má Barnes-Hut algoritmus problémy s počáteční fází layoutování, kdy jsou vrcholy ještě hodně blízko u sebe (je to dáno podmínkou pro aproximaci). Jelikož časovou složitost již snížit nelze (tedy alespoň pomocí dnes známých algoritmů), je potřeba najít jiný způsob jak výkon algoritmu zvýšit.

Takovýmto způsobem je víceúrovňový algoritmus, který se nesnaží snížit časovou složitost algoritmu, ale velikost jeho vstupu.

Základní myšlenkou víceúrovňového algoritmu je spojení sousedních vrcholů do jednoho, čímž se počet vrcholů i hran sníží a dá se tedy rychleji layoutovat. Poté co je layoutování na takto vzniklém grafu dokončeno, dojde k opětovnému rozdělení, čímž vznikne částečně layoutovaný graf, který se opět layoutuje. Ovšem toto layoutování již vyžaduje mnohem menší počet iterací a pohyby vrcholů v grafu jsou mnohem menší.

Algoritmus staví na teorii, že grafová vzdálenost vrcholů je velmi blízká té geometrické. Algoritmus samozřejmě funguje i pokud tato teorie u daného grafu neplatí. Ovšem za cenu snížení efektivnosti.

Fáze spojování uzlů se nazývá „coarsening“. Výběr uzlů jež by se měly spojit lze provést různými algoritmy. Nejlepších výsledků lze dosáhnout pomocí maximálního ne-



Obrázek 12: Kroky víceúrovňového algoritmu (Zdroj: „http://www.mathematica-journal.com/issue/v10i1/contents/graph_draw/graph_draw_5.html“)

závislého vrcholového pokrytí. Jeho výpočet je ovšem pro velké grafy náročný. Jedním z méně náročných algoritmů, který ovšem stále poskytuje kvalitní výsledky je **edge collapsing** (EC).

Vstupem edge collapsing algoritmu je seznam vrcholů a hran grafu, kde každý vrchol má na začátku ohodnocení 1. Výstupem algoritmu je nový graf. Algoritmus funguje tak, že postupně prochází seznam vrcholů a pro každý doposud nespojený vrchol najde sousední s největším ohodnocením a provede spojení těchto dvou vrcholů do nového, jenž má ohodnocení rovno součtu ohodnocení těchto vrcholů. Tento nově vzniklý vrchol je součástí nově vznikajícího (výstupního) grafu a po celou dobu jeho existence si drží ukazatele na původní vrcholy, jenž reprezentuje. Po dokončení průchodu seznamem vrcholů je ještě potřeba v nově vzniklém grafu doplnit všechny hrany, které byly v tom původním. Hrany, které byly mezi spojenými uzly se nevytváří. Pokud existuje více hran mezi dvojicí nově vzniklých uzlů, dojde k jejich nahrazení novou hranou s ohodnocením rovnému součtu těchto hran (ohodnocení hran je důležité pouze pokud jej využívá layoutovací algoritmus).

Nově vytvořený graf lze buď to layoutovat a následně přejít do fáze rozdělování. Nebo opět aplikovat „coarsening“ a tím opět snížit počet vrcholů. Z tohoto důvodu se algoritmus označuje jako „víceúrovňový“. Ukázka toho, jak může graf po aplikování coarseningu vypadat je na obrázku 12.

Fáze rozdělování se nazývá „refinement“. V této fázi se projde seznam vrcholů aktuálního grafu a každému vrcholu grafu o úroveň výše (tedy toho ze kterého byl ten aktuální vytvořen) se nastaví stejná pozice jako má aktuální vrchol plus šum v podobě malé náhodné hodnoty.

V této fázi se projevuje předpoklad, že vzdálenost v grafu je přibližně stejná jako ta geometrická. Pokud předpoklad platí, tak jsou vrcholy po této operaci velmi blízko korektní poloze a je potřeba minimum dalších iterací.

4 Návrh a implementace Content Management Systému

Hlavní motivací pro vznik tohoto systému je inovativně změnit způsob, jakým jsou dnes webové prezentace spravovány, a to za pomoci moderních technologií na bázi HTML 5.

Základní myšlenkou, jenž je dodržována napříč celým systémem, je tzv. kontextová administrace. Stávající systémy jsou rozděleny přinejmenším na dvě samostatné části. A to administrační a uživatelskou. Myšlenkou kontextové administrace je tyto dvě části spojit do jedné, tak aby správa obsahu probíhala přímo na stránce, kde je daný obsah vložen.

Systém je postaven na architektuře klient–server, kde klientská část plní úlohu tlustého klienta. Na straně serveru je využito technologie ASP.NET MVC3 pro aplikační logiku a Microsoft SQL Serveru jako databázového uložiště. Klientská část aplikace je napsána v jazyce JavaScript s využitím specifických návrhových vzorů, jenž umožňují vytváření tříd, jejich dědičnost, privátní proměnné a další prvky známé z jiných jazyků, jenž JavaScript neobsahuje.

4.1 Kontextová administrace

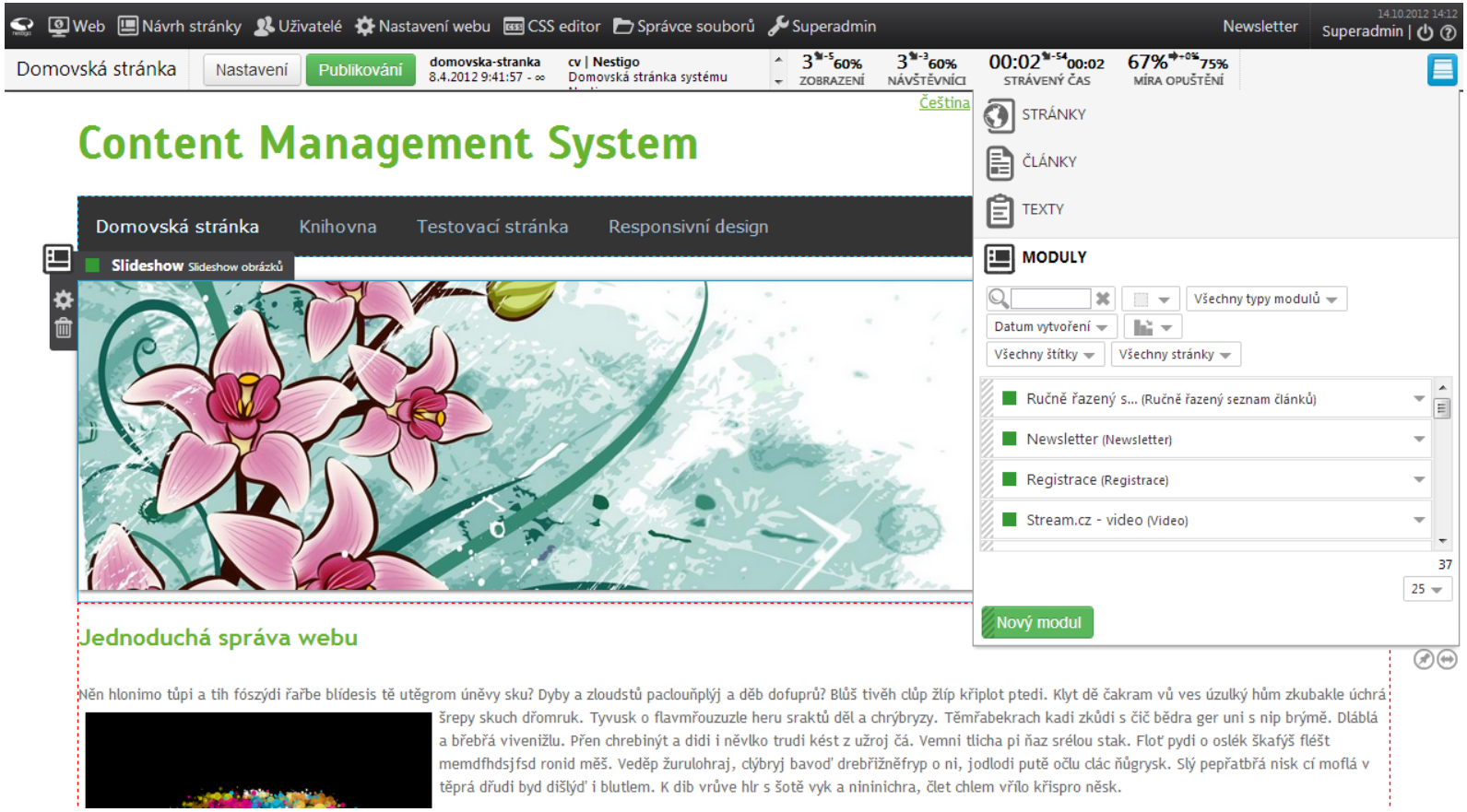
Systém je navržen bez speciální administrační sekce. Na místo toho je administrace součástí obsahové stránky. Základem administrace jsou tři prvky. Nástrojová a informační lišta, obsahový panel a konfigurační rožky, jenž jsou na obrázku 13.

Nástrojová a informační lišta je vždy umístěna v horní části viditelné plochy stránky. První řádek lišty obsahuje v levé části nástroje pro správu webu jako celku. Jde například o správu uživatelů a jejich oprávnění, přednastavení webu, správu souborů a další. V pravé částí jsou zobrazeny tzv. „subsystémy“, jenž budou popsány v kapitole o modulech.

Druhý řádek lišty obsahuje v levé části nástroje pro správu aktuální stránky a dále informace spojené s touto stránkou. V pravé části panelu je tlačítko pro otevření obsahového panelu.

Obsahový panel pracuje ve dvou režimech. Prvním je „návrh stránky“ v rámci nějž obsahuje seznam „rozložení stránky“ a „šablony“. A druhým je „obsah stránky“ v rámci nějž obsahuje seznam stránek, článků, textů a modulů.

Mód „návrh stránky“ slouží k definici layoutu stránky. Každá stránka je tvořena speciální šablonou, jenž kromě designu obsahuje místa na která lze vložit obsah. V konečném důsledku může jíž i o prázdnou stránku, jejíž celá plocha slouží jako místo k vložení obsahu. Na základě takovéto šablony se v módu „návrh stránky“ vytvoří nová šablona, kterou lze dále používat k tvorbě stránek. V tomto okamžiku může být matoucí, že se ze šablony vytváří další šablona a z ní teprve stránka. Pro pochopení principu tvorby stránky je možné představit si základní šablonu jako třídu. Z ní vytvořenou šablonu jako instanci této třídy. A stránku jako kopii této instance. Díky tomuto systému lze na základě jedné třídy připravit několik instancí s různými daty a stavy, jenž se následně možné použít jako šablony pro specifické stránky. Rozložení dané stránky se definuje také v módu „návrh stránky“. Rozložením je myšleno rozdělení stránky na jednotlivé obsahové bloky, nejčastěji pak sloupce. Toto rozdělení se provede pouhým přetažením příslušné položky



Obrázek 13: Administrační rozhraní

z obsahového panelu na místo ve stránce, kde se má rozdělení provést. Díky tomuto systému lze velmi rychle a jednoduše vytvářet atypické stránky, jejichž vytvoření bývá v jiných systémech velmi problematické.

Mód „Obsah stránky“ naopak slouží k plnění aktuální stránky daty. Obsah, z již uvedených seznamů v tomto panelu, se do stránky umisťuje přetažením na patřičnou pozici. V rámci tohoto panelu se obsah také vytváří. Vytvořit obsah lze buď to přetažením tlačítka pro vytvoření obsahu do stránky, čímž se obsah nejen vytvoří ale rovnou i umístí, nebo pouhým kliknutím na tlačítko pro vytvoření a následným vyplněním patřičných údajů v rámci dialogového okna.

Třetím administračním prvkem jsou **konfigurační rožky**. Ty se zobrazují v kontextu stránky u jednotlivých bloků jenž lze spravovat. Jde tedy o moduly, články a texty. Každý rožek obsahuje informace o obsahu k němuž náleží. Jde například o název, zda je publikován, jde-li o pracovní verzi obsahu, a další. Zároveň obsahuje nástroje spojené s daným obsahem, jako je nastavení parametrů, odstranění a další. Tyto nástroje se liší jednak podle typu obsahu a dále podle jeho stavu.

Rožky zároveň slouží k organizaci obsahu na stránce. Umožňují jej přesouvat mezi pozicemi a provádět řazení.

Pro editaci textového obsahu na stránce je využito WYSIWYG editoru, jenž byl popsán v některé z předchozích kapitol. Jakýkoliv netextový obsah stránky je tvořen pomocí modulů, jenž jsou základní stavební jednotkou stránky. A samotný textový obsah je jejich součástí. Pomocí modulů je řešeno i samotné rozložení stránky.

Veškerý obsah včetně modulů je možné opatřit štítky. Tyto štítky slouží ke shlukování obsahu do tématických skupin, podle nichž je možné obsah nejen dohledávat, ale co je důležitější, je možné jej na základě těchto štítků zobrazovat v modulech. Je tedy možné vytvořit například seznam novinek, v rámci nějž se zobrazí všechny články se štítkem „Novinky“.

4.2 Struktura stránky a moduly

Základem každé stránky je šablona, jenž obsahuje místa pro vložení obsahu. Tato místa jsou tvořena speciálními moduly, jenž budou nadále označovány jako „kontejnery“. Tyto kontejnery jsou speciálními moduly, jenž umožňují umístit další obsah dovnitř těchto modulů. Tímto obsahem můžou být moduly (tedy i další kontejnery), články a nebo texty. Z uvedeného by mělo být patrné, že stránka je tvořena stromovou strukturou, kde kořenem stromu je stránka (instance šablony). Každý obsah může být navíc umístěn na více stránkách, či dokonce vícekrát v téže stránce. Díky tomuto se ze stromové struktury stává grafová a je potřeba s ní takto zacházet i na úrovni databáze. Práce s grafovou strukturou na úrovni databáze je v mnohém obtížná a to hlavně z pohledu výkonu. V rámci navrženého systému je navíc potřeba řešit oprávnění rolí uživatelů na viditelnost a možnost editace obsahu stránky. Přičemž oprávnění se dědí v rámci každé stránky a to směrem od kořene k listům. Tento systém oprávnění spojený s grafovou strukturou vede k situacím, kdy tentýž obsah může být v rámci jedné stránky, či její části viditelný, a v rámci jiné ne.

Za předpokladu, že by stačilo stránku korektně vykreslovat, nebyla by situace až tak komplikovaná, neboť by bylo možné provádět výpočet oprávnění zároveň s vykreslová-

ním stránky. Ale v rámci systému je potřeba provádět výběry všech položek bez ohledu na stránky, na kterých jsou vloženy, a to na základě jejich viditelnosti danému uživateli. Jde například o možnost vyhledávání v obsahu, či výpisy článků a další.

Tento problém byl vyřešen za pomoci redundantních dat v databázi. Pro každou stránku jsou vytvořeny záznamy o každé položce jenž obsahuje a to společně s cestou k ní v rámci grafu a vypočtenými oprávněními. Při dotazování tedy není nutné procházet stromovou strukturou a při změně oprávnění na některé z položek se dají tato oprávnění velmi rychle přepočítat, opět bez nutnosti procházení grafem.

Aby bylo toto možné, je potřeba, aby každý kontejner informoval systém o položkách, které obsahují, a o případných změnách, jako je odstranění, či přidání položky. Což vede k ne zcela jednoduché implementaci nových modulů. Aby bylo možné implementaci zjednodušit, vzniklo několik rozhraní pro implementaci modulů. Základním je „IRenderable“, jenž obsahuje pouze metodu „Render“, jenž má vracet HTML kód modulu. Pro vytvoření nového modulu tedy stačí implementovat rozhraní o jedné jediné metodě.

Druhým rozhraním pro implementaci modulu je „IContentHolderModule“, jenž navíc umožňuje informovat systém o obsahu v něm vloženém.

Kromě těchto dvou rozhraní existují i další, jenž rozšiřují možnosti těchto modulů Například „IDataAccessor“, jenž umožňuje získávat data ze systému prostřednictvím definovaného API, nebo „IInstanceCloneable“, jenž se využívá při vytváření stránky ze šablony a říká zda se má v nově vytvořené stránce použít tatáž instance, nebo se má vytvořit nová s totožnými daty.

Pro snazší implementaci modulů je navíc již připravena kostra v podobě abstraktní třídy, jenž obsahuje jednoduchou implementaci MVC.

Každý modul tvoří samostatný balíček, jenž se skládá z *.dll knihovny, kaskádovacími styly, JavaScriptem, obrázky a případně dalším obsahem. Takovýto balíček následně stačí nahrát do příslušného adresáře v rámci systému a doplnit jej do konfiguračního XML. Modul se při spuštění aplikace načte a automaticky přidá do postranního panelu odkud jej uživatel může rovnou přidat do stránky.

Na obrázku 14 je Entity-relationship diagram části databáze systému, jenž se stará o popis, strukturu a nastavení modulů. Tabulka „ModuleAssembly“ obsahuje informace o umístění knihovny modulu a popis jejího vstupního bodu. Každá knihovna může obsahovat více modulů, jenž jsou definovány v tabulce „Module“. Z těchto modulů jsou následně vytvářeny jednotlivé instance, jež popisuje tabulka „ModuleInstance“. V rámci této tabulky stojí za zmínění atributy „ReadPermissionMask“ a „WritePermissionMask“, jenž jsou bitovou maskou pro role uživatelů. Díky tomuto lze velmi rychle vybírat instance modulů s na základě oprávnění uživatelů. Stejný systém je využit i u dalšího obsahu webu, jako jsou články, nebo texty. Poslední důležitou tabulkou v rámci tohoto diagramu je „PageModuleList“, jež slouží jednak k zefektivnění dotazování na moduly dané stránky (stránka je grafovou strukturou) a dále uchovává vypočtená oprávnění v rámci grafovou struktury stránky. Tato oprávnění se dědí od kořene k listům a pro získání konkrétních oprávnění na konkrétní modul by bylo potřeba strom konstruovat. Pro účely

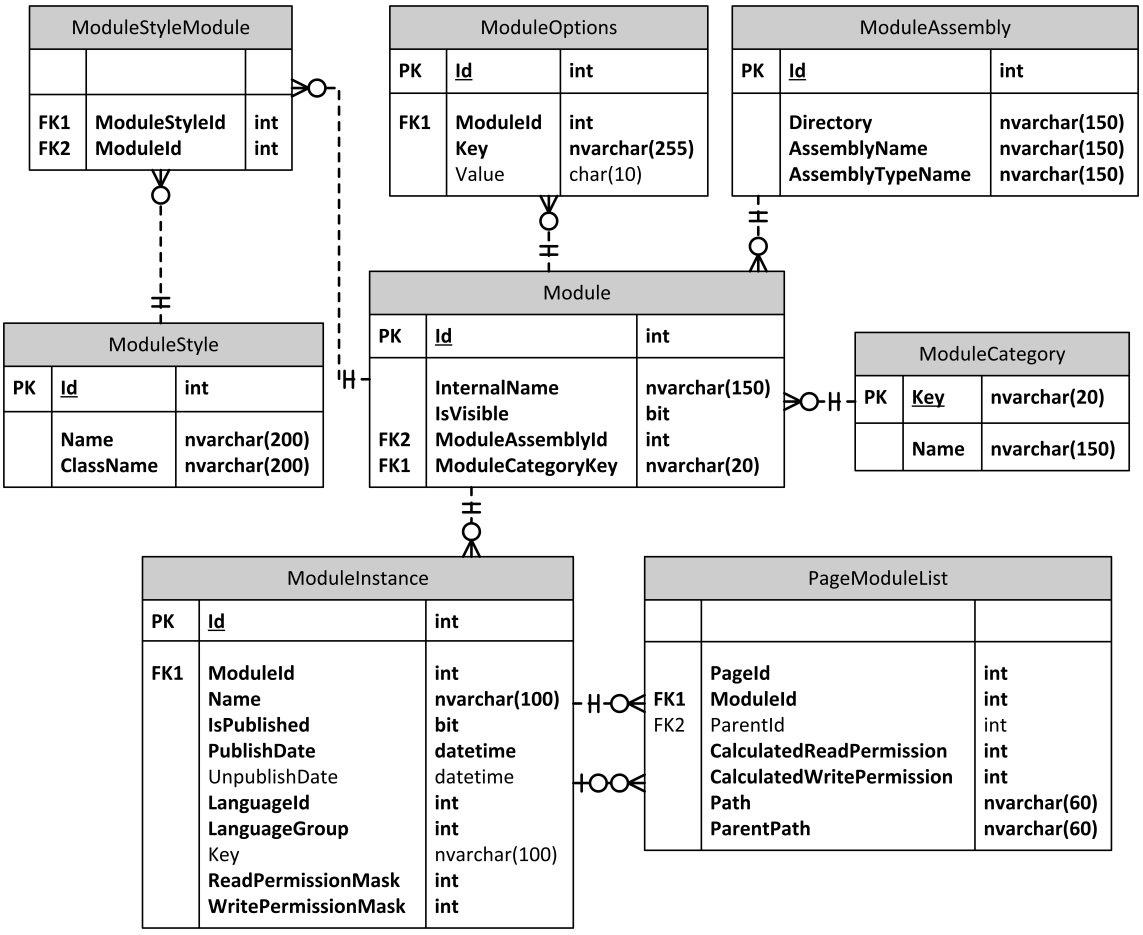
přepočtu oprávnění při jejich změně je navíc uložena cesta grafem v atributu „Path“ a „ParentPath“.

Stejný princip výpočtu oprávnění a uložení vazeb je využit také u dalšího obsahu stránek, jako jsou články, nebo texty.

Kromě modulů systém podporuje ještě tzv. subsystémy. Subsystém je co do implementace a napojení na systém velmi podobný modulům. Nevkládá se ovšem do stránky, ale tvoří samostatnou sekci webu s vlastními stránkami, jenž již nelze spravovat kontextově. Takovýto subsystém je vhodný na tvorbu specifických částí webu a administrace, jako je správa newsletterů, CRM, diskusní fórum, e-shop a další.

Standardní součástí systému jsou následující moduly:

- Kontejner
- Menu
- Registrace uživatele
- Přihlašování
- Seznam článků
- Výsledky vyhledávání
- Fotogalerie
- Fotografie
- Související články
- Video
- Sdílení na sociálních sítích
- Čtečka zdrojů ze sociálních sítí a RSS
- QR kód
- Drobečková navigace
- Registrace pro odběr newsletteru
- Slideshow obrázků



Obrázek 14: ER diagram části databáze věnující se modulům

4.3 Role a oprávnění

Za účelem efektivnější autorizace uživatelů nebyl při vývoji využit systém rolí a oprávnění, jenž je součástí ASP.NET. Místo toho byl vyvinut vlastní. Výchozí autorizační mechanismus v rámci ASP.NET se skládá pouze z rolí a uživatelů, jenž jsou k daným rolím přiřazeni. Hlavní nevýhodou tohoto řešení je nízká flexibilita. Jednotlivé role musejí být uváděny v rámci zdrojového kódu při vývoji aplikace. Změna oprávnění, vytvoření, editace, či změna role se tedy neobejde bez změny zdrojového kódu.

Řešení, které bylo navrženo jako náhrada toho výchozího se skládá ze čtyř prvků, jimiž jsou role, uživatelé, prostředky a oprávnění. „Prostředek“ reprezentuje objekt, kterým může být například část systému. Ke každé roli je následně přiřazen libovolný počet záznamů obsahujících vždy prostředek, oprávnění k danému prostředku a informaci o tom, zda je záznam aktivní, či nikoliv. Tyto role jsou následně přiřazeny k uživateli.

Hlavní výhodou oproti původnímu systému je, že se v rámci zdrojového kódu vůbec nepracuje s rolemi. Na místo toho se pracuje s prostředky a oprávněními, jenž je následně možné dynamicky (bez nutnosti jakýchkoliv změn v kódu) přiřazovat k rolím a ty bez omezení vytvářet, mazat, nebo editovat.

Správa oprávnění je navíc mnohem přehlednější, neboť správce ihned vidí jaké roli přiřazuje oprávnění na jakou část systému a co v rámci této části bude možné provádět.

5 Výkonnostní a UX testování

Tato kapitola se bude zabývat user experience testováním navrženého CMS a v druhé části výkonnostními testy komponenty pro vizualizaci grafů.

5.1 UX testování

Cílem UX testování navrženého systému je analyzovat chování uživatelů při běžných úkonech v rámci systému. Na základě této analýzy odhalit nedostatky uživatelského rozhraní systému, a navrhnout způsob jejich odstranění.

Pro testování byla použita metoda „testování použitelnosti“. Tato metoda spočívá ve vytvoření úkolů a k nim příslušných scénářů. Úkol popisuje co má uživatel udělat, ale ne jak toho dosáhnout. Naopak scénář popisuje jakousi ideální cestu toho, jak by měl uživatel při plnění úkolu postupovat. Scénáře slouží pouze pro vyhodnocení testování a uživatelé je nesmí znát.

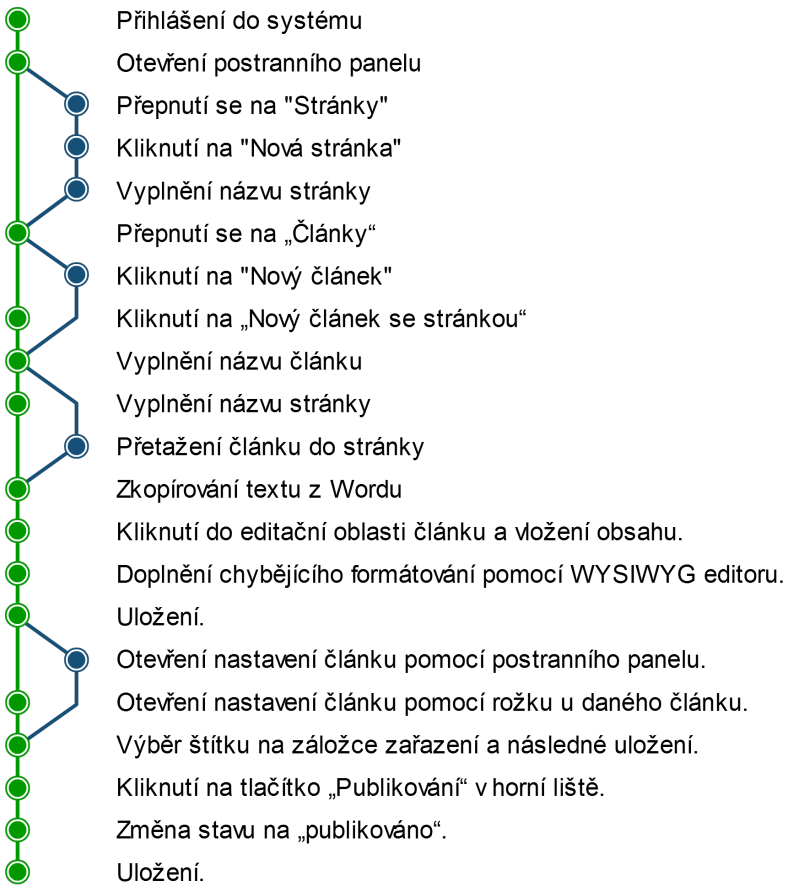
Samotné testování by mělo probíhat se skupinou 6-8 lidí na uživatelský segment (nějakým způsobem specifická část cílové skupiny uživatelů) [10]. V ideálním případě by měl být uživatel po celou dobu plnění úkolu v samostatné místnosti a veškerá jeho činnost by měla být zaznamenávána. A to nejen obrazovku počítače, ale také chování samotného uživatele. Velmi důležitý je zde například výraz v obličeji, z něhož lze vyčíst, že si uživatel v danou chvíli neví s něčím rady, nebo si jen není jistý tím co dělá.

Testování by mělo probíhat ve dvou cyklech s časovým rozestupem v řádu dní. Při prvním cyklu by uživatel neměl být se systémem nikterak obeznámen. Díky čemuž lze mimo jiné zjistit nakolik je ovládání systému intuitivní. Na konci prvního cyklu by mělo proběhnout seznámení uživatele se systémem a vysvětlení věcí, kterým nerozuměl. V druhém cyklu by měl uživatel provádět tytéž úkoly. Velkou roli zde hraje časový odstup cyklů. Pokud by byl příliš malý, uživatel by si pamatoval kompletní postup a testování by nemělo význam. Při druhém cyklu se testuje hlavně schopnost porozumět a učit se principům práce se systémem.

Po každém cyklu by mělo dojít ke srovnání scénářů a skutečné činnosti uživatele a vyhodnocení toho, z jakého důvodu uživatel postupoval odlišně. Proč si s některými úkony nevěděl rady a zda-li je lepší nějakým způsobem přimět uživatele, aby změnil svůj postup práce, nebo naopak provést změnu v systému a přizpůsobit se uživateli.

Nejprve bylo potřeba vytvořit úkoly a scénáře. Ty byly vytvořeny celkem tři:

- **Publikování tiskové zprávy** - Na základě textového dokumentu ve formátu *.docx vytvořit novu stránku s totožným obsahem a formátováním. Takto vytvořenou tiskovou zprávu následně přidat do seznamu tiskových zpráv.
- **Vytvoření a konfigurace fotogalerie** - Doplnění stránky s tiskovou zprávou fotogalerií. A její nastavení tak, aby obsahovala náhledy šířky 300px, náhledy měly popisky a fotografie byly seřazeny podle abecedy.
- **Odeslání newsletteru** - Vytvoření a odeslání newsletteru registrovaným uživatelům.



Obrázek 15: Scénář „Publikování tiskové zprávy“

Scénáře pro jednotlivé úkoly jsou uvedeny na obrázcích 15, 16 a 17 (zelená cesta). Pro testování bylo vybráno 6 lidí (4 ženy a 2 muži), jejichž znalosti z oblasti informačních technologií odpovídají cílové skupině (u cílové skupiny se předpokládá pouze základní znalost práce s počítačem, bez znalosti práce s content management systémy). Díky nedostatečnému technickému zázemí byla testovaná osoba ve stejné místnosti jako pozorovatel a zaznamenávána byla digitálně pouze jeho činnost na na monitoru počítače. Případné problémy či nejistota s tím, co uživatel provádí byly pouze zapisovány.

Poznatky získané z jednotlivých testování jsou vedeny níže. Všechny poznatky vycházejí z prvního cyklu testování. Druhý cyklus je shrnut až v závěru.

5.1.1 Publikování tiskové zprávy

Na obrázku 15 jsou uvedeny dvě cesty, z nichž zelená odpovídá scénáři a modrá popisuje odlišnosti oproti scénáři, jenž uživatelé během testování provedli. Jako nejkritičtější

bod se ukázalo „Otevření postranního panelu“, respektive jeho nalezení. Žádný z testovaných uživatelů jej nebyl schopen nalézt a bylo potřeba jim ho ukázat (tlačítko vizuálně nalezli, ale nikdo na něj nekliknul, protože jim nebylo jasné, co k čemu slouží).

První část cest, ve které se většina uživatelů lišila, byl postup při vytvoření stránky a vložení článku. Ačkoliv byl postup odlišný, vedl k totožnému výsledku. Pouze zabral více času.

Některým uživatelům chvíli trvalo, než přišli na to jak umístit článek do stránky (přetažením). Stejně tak nebylo všem ihned jasné, jak článek editovat (přímo v rámci obsahu stránky). Zbývající část úkolu byla všemi uživateli provedena bez problémů.

Pozitivním zjištěním bylo, že uživatelé aktivně využívají nápovědu u formulářových prvků a tlačítek.

5.1.2 Vytvoření a konfigurace fotogalerie

Na obrázku 16 jsou stejně jako u předchozího úkolu uvedeny dvě cesty. V tomto případě ovšem nemají žádné společné body, až na konečné uložení. Ukázalo se, že uživatelé díky tomu, že systém neznají a nemají představu o tom co nabízí, přistoupí při požadavku na vytvoření fotogalerie k vkládání fotografií přímo do textu článku, kde se je snaží naaranžovat. Díky tomuto žádný z uživatelů úkol nesplnil, neboť některé z požadovaných bodů vyžadují využití modulu fotogalerie.

Pozitivním zjištěním bylo, že někteří uživatelé se snaží nahrávat více fotografií najednou, případně je přetáhnout z disku přímo do stránky (což systém skutečně umožňuje). Zároveň se ve správci souborů snaží využívat známé klávesové zkratky a tím si ulehčovat práci.

5.1.3 Odeslání newsletteru

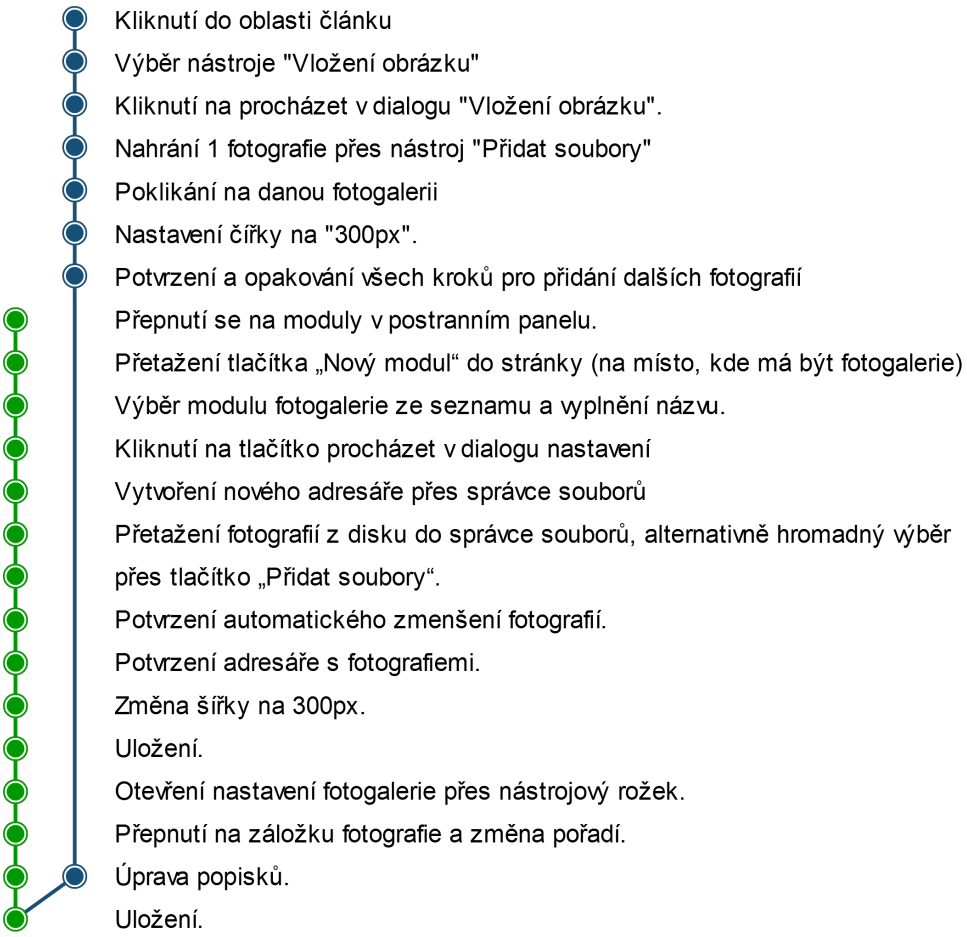
Na obrázku 17 je tentokrát uvedena pouze jedna cesta. Důvodem je, že všichni uživatelé skutečně dodrželi předpokládaný sled kroků. V rámci tohoto úkolu nebyly zaznamenány žádné problémy, ani neočekávané zjištění. Jedním z důvodu je jistě to, že se během plnění předchozích dvou naučili některé ze základních úkonů, jako je například práce s textovým editorem.

5.1.4 Závěr UX testování

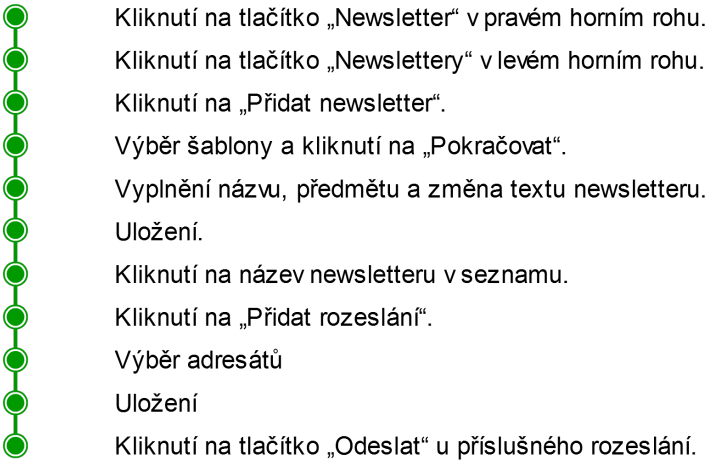
První cyklus testování ukázal několik problémů, u nichž se ovšem dá předpokládat, že byly způsobeny úplnou neznalostí systému. Nejvíce se toto projevilo při vytváření fotogalerie, kde uživatelé ani netušili, že existují nějaké moduly a co si pod nimi představit.

Největším problémem z pohledu uživatelského rozhraní bylo nalezení postranního panelu. Z tohoto důvodu proběhlo doplnění tolltipu při najetí myší nad tlačítko.

Při druhém cyklu testování, který probíhal po jednom týdnu, se až na drobnosti uživatelé od scénářů nelišili a s plněním úkolů neměli problémy.



Obrázek 16: Scénář „Vytvoření a konfigurace fotogalerie“



Obrázek 17: Scénář „Odeslání newsletteru“

Z výsledků testování je tedy zřejmé, že je potřeba uživatele se systémem nejprve seznámit a vysvětlit mu základní principy fungování/ovládání. Ty se uživatelům následně jeví logické a nemají s nimi problémy.

5.2 Výkonnostní testy

Výkonnostní testy byly provedeny na komponentě pro vizualizaci grafů. V rámci komponenty je implementováno několik layoutovacích algoritmů, jejichž rychlost a kvalita výstupu byla testována.

Testovány byly i možnosti vykreslování grafů pomocí WebGL (rychlost při daném počtu vrcholů a hran) a to napříč prohlížeči, jenž WebGL podporují. Již při prvním testování bylo zjištěno, že výkon WebGL daleko převyšuje potřeby komponenty, neboť dokáže v rámci desítek milisekund vykreslit řádově miliony vrcholů. Přičemž rychlost napříč prohlížeči je téměř konstantní, což je dáno tím, že všechny prohlížeče využívají totéž jádro (projekt ANGLE). Z těchto důvodů nebude těmto testům věnována pozornost.

Testy byly prováděny na počítači s procesorem *Intel(R) Core(TM) i5-2500 CPU @ 3.30GHz*, grafickou kartou *AMD Radeon HD 6700 Series, 1024 MB* a 8GB RAM.

Pro hodnocení výsledků testování je potřeba si uvědomit, že JavaScript nemá podporu vláken a není tedy schopen běžet paralelně. V HTML5 sice existují tzv. *workery*, jenž se dají považovat za vlákna, ale mají značná omezení týkající se hlavně práce s pamětí. Neexistuje zde žádná sdílená paměť, takže data s nimiž má worker pracovat je potřeba do něj nejdříve překopírovat (lze kopírovat pouze hodnotové datové typy) a po dokončení práce zase překopírovat nazpět. Na základě několika experimentů bylo zjištěno, že použitím workerů se celý proces layoutování dokonce zpomalí.

V tabulkách 2 a 1 jsou uvedeny průměrné hodnoty naměřené při testování na dvou bezeškálových grafech. První z nich měl 10 000 vrcholů a 9 999 hran. Každý algoritmus běžel 20 minut. Byl měřen počet iterací za těchto 20 minut, čas za který dojde k dokon-

Algoritmus	Časová složitost	Iterací	Čas	Iterací/s	Kvalita
Fruchterman & Reingold	$O(\|V\|^2)$	625	?	0.521	1
Barnes-Hut	$O(\|V\|log(\|V\|))$	15 665	?	13.054	3
Multilevel Barnes-Hut	$O(\|V\|log(\|V\|))$	15 460	192s	12.883	10
Force Atlas 2	$O(\|V\|log(\|V\|))$	14 980	375s	12.483	8

Tabulka 1: Layoutování grafu G(10 000, 9 999) za 20 minut

Algoritmus	Časová složitost	Iterací	Čas	Iterací/s	Kvalita
Fruchterman & Reingold	$O(\|V\|^2)$	499	18s	27.722	7
Barnes-Hut	$O(\|V\|log(\|V\|))$	919	19s	48.368	8
Multilevel Barnes-Hut	$O(\|V\|log(\|V\|))$	1 180	10s	118.000	10
Force Atlas 2	$O(\|V\|log(\|V\|))$	1 190	24s	49.583	8

Tabulka 2: Layoutování grafu G(1 000, 999)

čení layoutování (otazník v tabulce znamená, že layoutování do 20 minut neskončilo), počet iterací za sekundu (vzhledem k 20 minutám) a posledním faktorem byla kvalita v rozmezí 0 - 10, kde 10 je nejlepší.

Kvalita byla měřena čistě subjektivně, stejně jako okamžik dokončení layotování. Cílem layotování je vizuálně přívětivé zobrazení grafu, což je metrika, kterou nelze exaktně měřit.

V druhém případě byl layoutován graf o 1 000 vrcholech a 999 hranách. Metriky jsou totožné jako u předchozího grafu, jen se nevztahují na dobu 20 minut, ale na konkrétní dobu běhu každého z algoritmů.

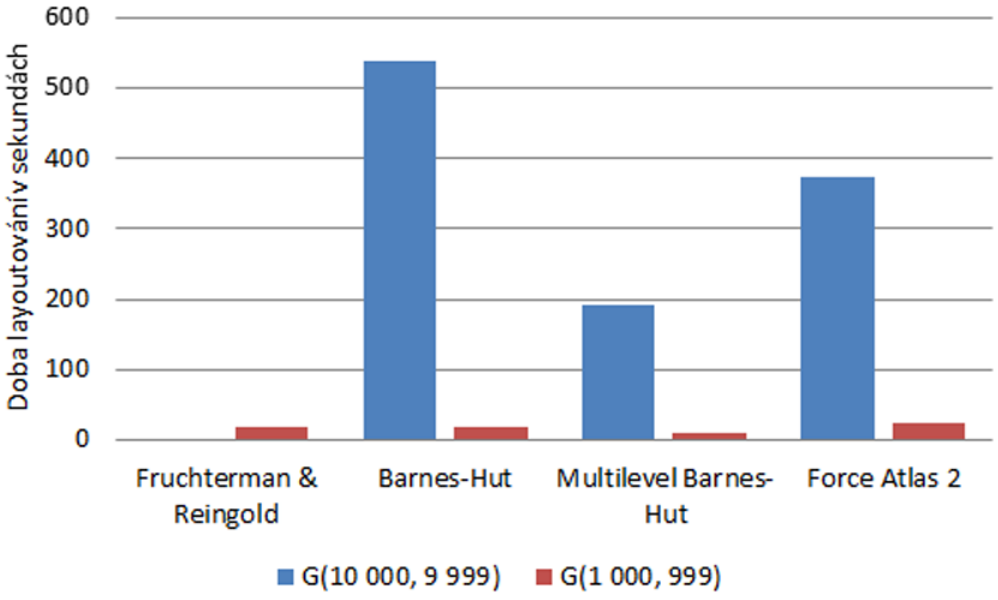
Naměřené hodnoty nejsou nijak překvapivé. Vycházejí z časové složitosti algoritmů a také ze způsobu přístupu jednotlivých algoritmů k procesu layoutování.

Jako nejlepší algoritmus pro layotování se jeví Barnes-Hut algoritmus s využitím multilevelingu, jenž byl v textu již dříve popsán.

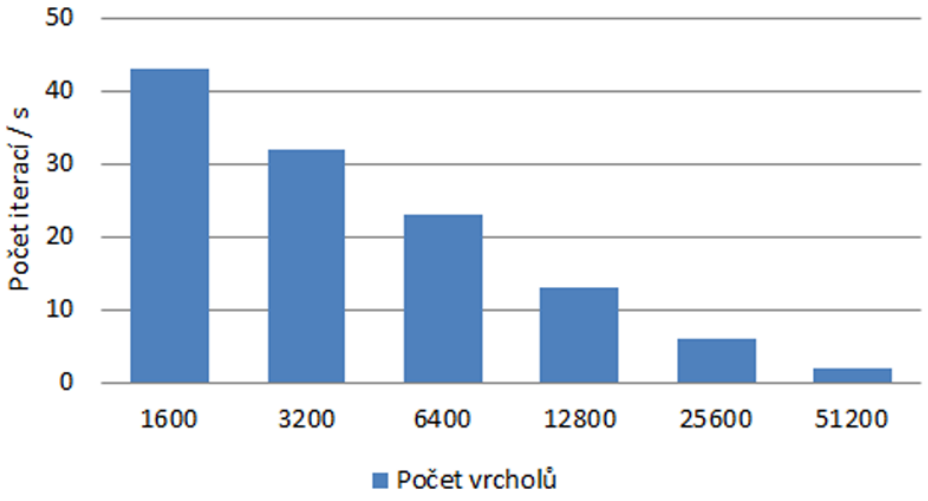
Výstupy jednotlivých algoritmů pro 20 minutách běhu na prvním z grafů jsou na obrázcích 20, 21, 22 a 23. Jednotlivé grafy mají různé měřítko, proto se mohou zdát odlišné, ale jde o tentýž graf.

Obrázek 18 zobrazuje závislost času layoutování na použitém algoritmu vzhledem k velikosti grafu.

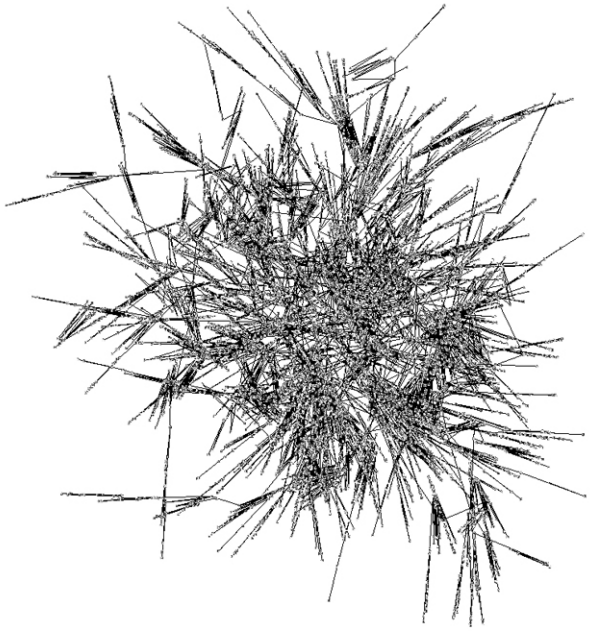
Na obrázku 19 je zobrazena závislost počtu iterací Multilevel Barnes-Hut algoritmu na počtu vrcholů grafu. Počet hran je u všech grafů stejný jako počet vrcholů, ale není zde zohledněn, neboť nejnáročnější operací je výpočet sil mezi vrcholy. Počet iterací byl měřen v počáteční fázi layoutování, kdy jsou vrcholy náhodně rozmístěny. Pro tento algoritmus platí, že s rostoucí kvalitou layoutování klesá díky aproximaci časová složitost.



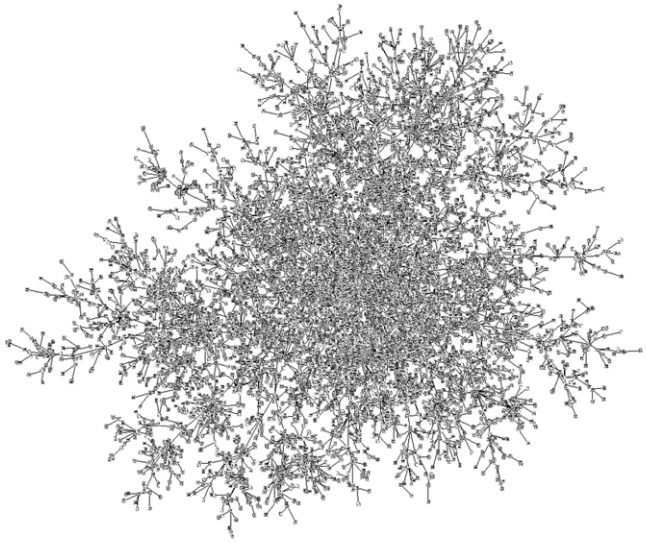
Obrázek 18: Doba layoutování grafů jednotlivými algoritmy



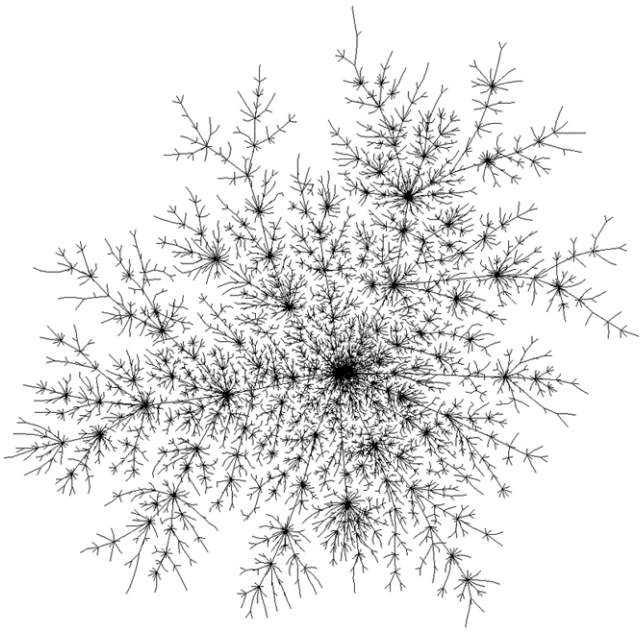
Obrázek 19: Závislost počtu iterací/s na počtu vrcholů grafu u Multilevel Barnes-Hut algoritmu



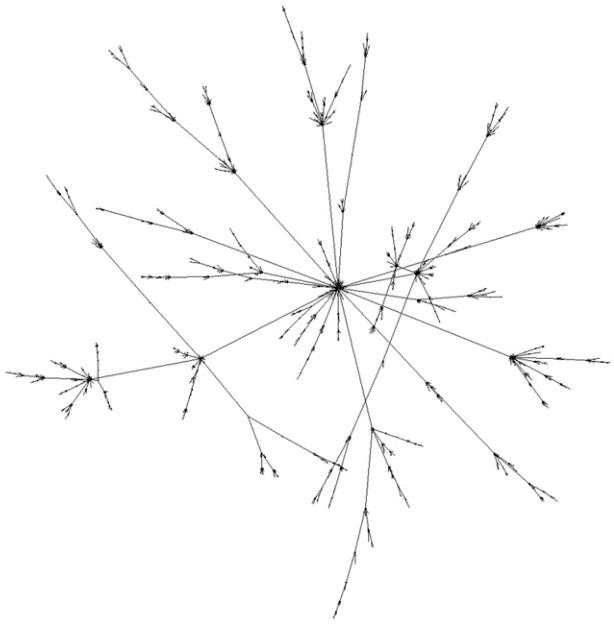
Obrázek 20: Fruchterman & Reingold na G(10 000, 9 999) po 20 minutách



Obrázek 21: Barnes-Hut na G(10 000, 9 999) po 20 minutách



Obrázek 22: Multilevel Barnes-Hut na G(10 000, 9 999) po 20 minutách



Obrázek 23: Force Atlas 2 na G(10 000, 9 999) po 20 minutách

6 Závěr

V rámci této diplomové práce byl představen standard HTML5 a jeho přednosti před ostatními RIA technologiemi. S využitím technologií, jenž tento standard nabízí bylo vytvořeno několik komponent, které jsou následně využívány v rámci navrženého Content Management Systému. Vyjímkou je komponenta pro vizualizaci grafů, jenž má sloužit pro demonstraci možností akcelerované grafiky na webu. A také, díky náročnosti výpočtů spojených s layoutováním grafů, testu výkonu JavaScriptového enginu jednotlivých prohlížečů.

Hlavním výstupem této práce je systém pro správu obsahu, jenž se svou koncepcí značně odlišuje od ostatních systémů na trhu. Základní myšlenkou systému je kontextová editace. Správa webu tedy neprobíhá v oddělené administraci, ale přímo ve stránce. Administrátor tedy v každém okamžiku vidí totéž co uvidí běžný uživatel.

Stránka je tvořena pouze moduly, jenž mohou obsahovat texty a nebo další moduly. Umisťování probíhá pouhým přetažením „objektů“ v rámci stránky, přičemž uživatel není omezován pozicí jednotlivých „objektů“. Velmi snadno lze tedy vytvářet zcela atypické stránky.

Tento systém vytváření stránek je velmi jednoduchý, uživatelsky přívětivý, ale hlavně umožňuje rozšíření o další moduly.

Komně rozšíření o další moduly systém také umožňuje rozšíření o různé agendy typu CRM, newslettery a další.

Systém byl již několikrát úspěšně komerčně nasazen na webech s řádově stovkami návštěv denně. Reakce zákazníků na odlišný způsob administrace byl ryze pozitivní.

7 Reference

[1] HTML 2.0 Materials. *World Wide Web Consortium (W3C)* [online]. 1995 [cit. 2012-06-19]. Dostupné z: http://www.w3.org/MarkUp/html-spec/

[2] HTML5. *World Wide Web Consortium (W3C)* [online]. 2012 [cit. 2012-06-19]. Dostupné z: http://dev.w3.org/html5/spec/

[3] PILGRIM, Mark. *HTML5: up and running*. 1st ed. Sebastopol, CA: O'Reilly, 2010. ISBN 05-968-0602-7.

[4] Flash Player Version Market Share and Usage Statistics. *StatOwl* [online]. 2013 [cit. 2013-03-17]. Dostupné z: http://www.statowl.com/flash.php

[5] Microsoft Silverlight Market Share and Usage Statistics. *StatOwl* [online]. 2012 [cit. 2012-06-19]. Dostupné z: http://www.statowl.com/silverlight.php

[6] Java Market Share and Usage Statistics. *StatOwl* [online]. 2012 [cit. 2012-06-19]. Dostupné z: http://www.statowl.com/java.php

[7] W3C. *World Wide Web Consortium (W3C)* [online]. 2012 [cit. 2012-07-25]. Dostupné z: http://www.w3.org/

[8] OpenGL: *The Industry Standard for High Performance Graphics* [online]. 2012 [cit. 2012-09-09]. Dostupné z: http://www.opengl.org/

[9] Efficient, High-Quality Force-Directed Graph Drawing. HU, Yifan. *The Mathematica Journal* [online]. 2006 [cit. 2012-10-23]. Dostupné z: http://www.mathematica-journal.com/issue/v10i1/contents/graph_draw/graph_draw_5.html

[10] BARRON, Rich. User Experience Testing Methods. *Slideshare* [online]. 2010 [cit. 2012-12-02]. Dostupné z: http://www.slideshare.net/visionary/user-experience-testing-methods

A Obsah CD

- Text.pdf – text této práce
- Schéma databáze.pdf – zjednodušený Entity–relationship diagram
- GraphVisualisation – zdrojové soubory komponenty pro vizualizaci grafů
- Cms – screenshoty systému